Veikko Salminen

Virtuaalinen tietokanta sovellusalustana

Elektroniikan ja sähkötekniikan koulutusohjelma

Diplomityö

Espoo 6.3.2014

Valvoja:

Prof. Eero Hyvönen

Ohjaaja:

DI Antti Tölli

Aalto-yliopisto
Sähkötekniikan korkeakoulu

AALTO YLIOPISTO

DIPLOMITYÖN TIIVISTELMÄ

Tekijä: Veikko Salminen

Työn nimi: Virtuaalinen tietokanta sovellusalustana

Päivämäärä: 6.3.2014 Kieli: Suomi Sivumäärä: 57+7

Tutkinto-ohjelma: Elektroniikka ja sähkötekniikka

Valvoja: Prof. Eero Hyvönen

Ohjaaja: DI Antti Tölli

Tässä työssä tarkastellaan Movenium Oy:n toteuttamaa virtuaalista tietokantaratkaisua nimeltä Collector. Työn alussa esitellään tarvittava teoria tietokannoista ja SaaS-palveluista. Kirjallisuusosuudessa verrataan virtuaalista tietokantaratkaisua eri tietokantoihin, kuten perinteiseen relaatiomalliin sekä NoSql-tietokantoihin.

Tutkimuksen aiheena on Collectorin kuormankestokapasiteetti ja sen riittävyys Movenium Oy:n tarpeisiin tulevaisuudessa. Tutkimus on toteutettu ohjelmalla, joka kasvattaa tietokantaa satunnaisdatalla ja suorittaa hakuja. Hakujen suoritusaikojen perusteella voidaan päätellä tietokantaratkaisun kuormankestävyys.

Avainsanat: Tietokanta, SaaS, MySql, pilvipalvelu

AALTO
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE
BACHELOR'S THESIS

Author: Veikko Salminen

Title: Virtual Database as a Development Platform

Date: 6.3.2014 Language: Finnish Number of pages: 57+7

Degree programme: Electronics and electrical engineering

Supervisor: Prof. Eero Hyvönen

Advisor: M.Sc. (Tech.) Antti Tölli

This bachelor's thesis is about virtual database solution called Collector. First section describes the needed theory about databases and SaaS. After theory part, Collector is being evaluated against different kind of database solutions. Both relational databases and NoSql-databases are covered.

Main focus of the research is to review Collector's performance and capacity. This is done by a script which writes data to database and executes search queries to determine how long does the searches take.

Keywords: Database, SaaS, MySql, Cloud Service

Esipuhe

Haluan kiittää työni valvojaa professori Eero Hyvöstä rakentavasta palautteesta ja työni tekemistä edistävistä ohjeista.

Kiitän myös ohjaajani Antti Tölliä, joka näki suuren vaivan työni arvioinnissa ja oikolukemisessa.

Kiitos vaimolleni, joka mahdollisti minulle kirjoitusaikaa työtäni varten ja kannusti työn loppuunsaattamisessa.

Mäntsälä, 6.3.2014

Veikko Salminen

Sisällys

Esipuhe iv
Sisällys v
Lyhenteet vii
1 Johdanto 1
2 SaaS-palvelut ja niiden tietokantaratkaisut 5
2.1 SaaS-palvelut 5
2.1.1 SaaS-palveluiden tuotekehitys 6
2.1.2 Tietoturva 6
2.2 Pilvipalvelualustat 6
2.3 Tietokanta 7
2.3.1 Tietokantataulut 8
2.3.2 Tietokantataulujen yhdistäminen kyselyissä 9
2.3.3 Indeksointi 9
2.3.4 Skaalautuvuus 10
2.3.5 Asiakkaiden datan hallinta 12
2.4 Movenium Collector 13
3 Tietokantaratkaisut ja kehitysmenetelmät kirjallisuudessa 15
3.1 Relaatiotietokannat 15
3.2 NoSql 18
3.2.1 Käytetyimmät NoSql-tietokannat 18
3.2.2 NoSql-tietokantojen suorituskyky 19
3.2.3 NoSql ja tietoturva 21
3.2.4 NoSql:n käyttäminen työajanseuranta palvelussa 22
3.3 Tietokanta palveluna (DBaaS) 22

3.4 Ruby on Rails 23
3.5 Päätelaitesovellukset 24
4 Collectorin kuormankestävyyden tutkiminen 28
4.1 Tutkimusmenetelmät 28
4.1.1 Tutkimuksessa käytetyt laitteet ja ohjelmistot 29
4.1.2 Tietokannan muoto 29
4.1.3 Tiedon kirjoittaminen tietokantaan 29
4.1.4 SQL-kyselyt 30
4.1.5 Palvelun testaaminen suuressa tietokannassa 32
4.1.6 Testien suorittaminen 33
4.1.7 Tulosten vertaaminen tuotantotietokantaan 33
4.1.8 Tuotantotietokannan kasvunopeuden arviointi 34
4.2 Tulokset ja niiden arviointi 36
4.2.1 Hitaiden tietokantahakujen paikallistaminen 38
4.2.2 Tuotantotietokannan kopiolla suoritetut kontrollihaut 40
4.2.3 Tuotantotietokannan kasvunopeuden arvioinnin tulokset 43
5 Yhteenveto 45
5.1 Vastaukset tutkimuskysymyksiin 48
5.2 Jatkotutkimustarpeet 49
Viitteet 51
Liite: ohjelmakoodi tietokannan tutkimiseen 57

Lyhenteet

DOM	Document Object Model. Dokumentin puurakenne.
HTML	Hypertext Markup Language. Hypertekstin merkintäkieli.
JSON	JavaScript Object Notation. Yksinkertainen tiedonsiirtoformaatti.
MD5	message-digest -algoritmi. Tiivistysalgoritmi.
MVCC	Multiversion concurrency control. Tietokantatransaktioiden muistitekniikka.
MVC	Model-View-Control. Ohjelmistoarkkitehtuurityyli.
PHP	Hypertext Preprocessor. Komentosarjakieli web-palveluiden toteuttamissen.
RESTful	Representational state transfer. Arkkitehtuurimalli rajapinnoille.
SaaS	Software as a Service. Ohjelmisto palveluna.
XML	Extensible Markup Language. Merkintäkieli.

1 Johdanto

Tämän tutkimuksen tarkoituksena on selvittää Movenium Oy:n toteuttaman virtuaalisen tietokantaratkaisun [1] soveltuvuus suurelle käyttäjämäärälle tarkoitetun työajanseurantapalvelun alustaksi. Ratkaisun kaupallinen nimi on Collector. Alustaa verrataan kirjallisuudessa esiintyviin muihin vaihtoehtoihin ja tutkimuksessa tutkitaan alustan kuormankestävyys tulevina vuosina tietomäärän lisääntyessä. Tutkimuskysymykset ovat seuraavat:

1. **Mitä tietokantaratkaisuja Movenium Oy:n tarpeisiin on tarjolla?**

Collectorin kehittäminen ja ylläpitäminen kuluttavat yrityksen resursseja. Tästä syystä yrityksen tulee olla tietoinen mitä muita vaihtoehtoja tietokantaratkaisuksi on olemassa. Vaihtoehtoisia ratkaisuja etsitään kirjallisuudesta ja niitä verrataan Collectorin ominaisuuksiin.

2. **Riittääkö Collectorin suorituskyky tulevina vuosina Movenium Oy:n tarpeisiin?**

Koska Collector on tällä hetkellä käytössä kaikissa yrityksen tuotteissa, on tärkeää tutkia riittääkö järjestelmän suorituskyky noin 3-5 vuoden aikajaksolla.

3. **Onko Collector paras valinta Movenium Oy:n tietokantaratkaisuksi?**

Movenium Oy:n itse kehittämä Collector on tällä hetkellä alustana kaikissa Movenium Oy:n tuotteissa. Yritykselle on tärkeä tieto selvittää onko Collectorin käyttäminen nyt ja tulevaisuudessa (lähivuosina) järkevää vai tulisiko yrityksen vaihtaa ratkaisua.

4. **Mitä muutoksia Movenium Oy:n olemassa oleviin tuotteisiin tarvitaan suorituskyvyn varmistamiseksi?**

Tuotantokäytössä olevien palveluiden suorituskyvyn mittaamisessa yksi osa alue on järjestelmän pullonkaulojen havaitseminen. Järjestelmässä voi olla kohtia, jotka eivät vielä vaikuta suorituskykyyn merkittävästi, mutta jotka tietomäärän kasvaessa voivat

hidastaa järjestelmän toimintaa. Nämä ongelmakohdat tulee paikantaa ja korjata ennen ongelmien ilmaantumista.

Nykypäivänä suuri osa tietokoneohjelmista toteutetaan SaaS-palveluina (Software as a Service) [2][3]. SaaS-palvelu on internet-selaimessa toimiva ohjelma, joka suoritetaan palveluntarjoajan palvelimella. Palveluun tallennettavat tiedot tallentuvat tietokantaan ja palvelu hakee sieltä tarvitsemansa datan reaaliaikaisesti käyttäjälle. Tietokannan toiminnan varmuus ja tehokkuus ovat ehkä tärkein osa koko palvelun toiminnan kannalta. Tietokannan tulee säilyttää kaikki sisältämänsä data virheettömänä ja palvelun tulee voida hakea dataa tietokannasta mahdollisimman pienellä viiveellä. Relaatiotietokantaa käytettäessä SaaS-palvelun kehittäjä joutuu käyttämään tietokantaa melko matalalla tasolla, eli suorittamaan tietokannassa kyselyitä ja muuttamaan tietokannalta haetun datan esimerkiksi internet-selaimen käyttämään formaattiin. Suuri osa tietokantaa koskevista tarpeista on täysin tai lähes identtisiä eri SaaS-palveluiden välillä. Tästä johtuen suuri osa eri SaaS-palveluiden toteutuksesta voitaisiin hoitaa samalla lähdekoodilla [4].

Movenium Oy tarjoaa yritysten työajanseurantaratkaisua[1] SaaS-palveluna. Työntekijät syöttävät työajat tietokoneella tai matkapuhelimella pääkäyttäjän hallinnoimaan järjestelmään, josta palkanlaskijat ja työnjohtajat voivat tarkastella ja muokata työaikoja tietokoneella. Tietokantaratkaisuna käytettiin aiemmin MySql-tietokantaa[2] ilman kunnollista arkkitehtuuria tietokannan ja palvelun välillä. Tästä aiheutui ongelmia kehitystyöhön. Palveluiden räätälöinnin tarve tuo haasteen työajanseurantapalvelun tietokantaratkaisulle. Lähes jokaisella yritysasiakkaalla on omat yksilölliset työajanseurannan tarpeet. Se tekee palvelun tuotteistamisesta ja tietokannan rakenteen suunnittelusta erityisen hankalaa. Kun jokaisen palvelun tietosisältö on erilainen, tarvitaan ratkaisu, jossa jokaisessa palvelussa on käytännössä oma tietokantansa. Tähän tarpeeseen suunniteltiin MySql-tietokannan päälle virtuaalinen

[1] Movenium Oy:n työajanseurantapalvelu. http://movenium.com/
[2] MySQL wikipediassa http://en.wikipedia.org/wiki/MySQL.

tietokanta Collector. Virtuaalisen tietokannan tehtävänä on hallita tietokannan rakenne sekä datan tallentaminen ja lukeminen.

Eräs esimerkki suositusta nopean ohjelmistokehityksen tekniikasta on Ruby on Rails [5]. Sen tarkoituksena on tarjota suurin osa SaaS-palvelun tarvitsemista tekniikoista valmiina. Tekniikka sisältää valmiin tuen tietokantarakenteen mallintamiseen ja käyttöliittymän toteuttamiseen. Ruby on Rails -tekniikalla palvelun tekeminen on helppoa ja nopeaa, mutta tälläkin tekniikalla palvelun sisältö ja ominaisuudet täytyy toteuttaa kiinteästi [6]. Tekniikka ei suoraan mahdollista palvelua, jota voitaisiin joustavasti muokata joka asiakkaalle erikseen.

Collectorin virtuaalisen tietokannan perusajatus taas on, että tietokannan sisällön muotoa ei rakenneta tietokantaan kiinteästi. Perinteistä relaatiotietokantaa käytettäessä kaiken tietokantaan tallennettavan datan muoto kuvataan kiinteästi tietokantaan [7]. Kiinteä malli soveltuu hyvin sellaisiin palveluihin ja järjestelmiin, joissa tarvittava tietomuoto on tarkasti määriteltävissä ja erot eri asiakkaiden tarpeissa ovat vähäisiä. Virtuaalisessa tietokannassa jokaisen asiakkaan lomakkeisto (vrt. perinteisen relaatiotietokannan tietokantataulurakenne) kuvataan tietokannassa tähän tarkoitukseen luotujen erillisten tietokantataulujen avulla. Itse data tallennetaan tietokantaan hajautetusti eri tietomuotoja sisältäviin kokoojatietokantatauluihin. Asiakkaan lomakkeet, niiden sisältämät kentät ja näihin kenttiin liittyvät asetukset lisätään tietokantaan XML-formaatilla (Extensible Markup Language) [8][9] toteutettujen ohjaustiedostojen avulla. Jokaista eri palvelua varten on kirjoitettu kyseisen palvelun ominaisuudet lisäävä ohjaustiedosto.

Palvelusta löytyy myös asiakkaan itse hallittavissa olevia työkaluja, joiden avulla asiakas voi itse rajoitetusti säätää oman palvelunsa lomakkeistoa. Asiakkaalla on palvelussaan valittavana lista valmiita asetuksia, joista jokaisen taustalla on samankaltainen XML-ohjaustiedosto kuin palveluillakin. Asiakkaan laittaessa jonkin asetuksen päälle tai pois, asetusta vastaava ohjaustiedosto säätää asiakkaan palvelun lomakkeistoa. Tämän lisäksi

asiakas voi lisätä joillekin lomakkeille itse erityyppisiä syöttökenttiä[3] täysin omien tarpeidensa mukaisesti. Kun Collectorista haetaan dataa, järjestelmä luo dynaamisesti kyseiselle lomakkeelle tarvittavan kyselyn.

Koska asiakas voi itse vaikuttaa keräämänsä datan muotoon hallitsemalla lomakkeitaan, täytyy myös raportoinnin mukautua automaattisesti datan muotoa vastaavaksi. Tähän tarkoitukseen Collector sisältää valmiin raportointikäyttöliittymän sekä valmiit työkalut, joiden avulla palvelun käyttäjä voi muokata työaikoja ja muita tietoja. Normaalisti lomakkeet ohjelmoidaan jokainen erikseen [10]. Collector luo lomakkeet automaattisesti tietokannassa kuvatun mallin mukaisesti.

Tässä työssä esitellään aluksi yleistä teoriaa. Ensimmäisessä osuudessa käsitellään SaaS-palveluiden ominaisuuksia ja käsitteitä, sekä erilaisten tietokantojen tekniikoita. Tämän jälkeen esitellään kirjallisuudesta löytyviä ratkaisuja SaaS-palveluiden toteutukseen ja tietokantaratkaisuksi. Kirjallisuudesta löytyviä tekniikoita verrataan Movenium Oy:n kehittämään Collectoriin. Tarkasteltavana on sekä relaatiomalliin pohjautuvia, että ilman relaatiomallia toimivia tietokantaratkaisuja. Osio käsittelee myös palveluna tarjottavia tietokantoja. Viimeisessä osuudessa tutkitaan Collectorin suorituskykyä suurilla tietomäärillä. Collectorin tietokantaan kirjoitetaan noin 100 Gt dataa ja tämän jälkeen suoritetaan ennalta suunniteltuja hakuja. Hakujen suoritusaikojen perusteella arvioidaan toteutuksen kuormankestävyyttä tietomäärien kasvaessa.

Tutkimuksen perusteella näyttäisi siltä, että Collectorin suorituskyky tulisi riittämään muutamilla korjauksilla seuraavien 3-5 vuoden ajan. Vaihtoehtoiset tietokantaratkaisut ovat kuitenkin skaalautuvuudeltaan parempia.

[3] Esittely työajanseurantapalvelun muokattavuudesta: http://movenium.com/features/customize-employee-time-card.

2 SaaS-palvelut ja niiden tietokantaratkaisut

Tässä kappaleessa käsitellään tarvittava tietokantoihin ja SaaS-palveluihin liittyvä teoria.

2.1 SaaS-palvelut

SaaS-palvelu (Software as a Service) on tietokoneohjelma, joka tarjotaan asiakkaalle valmiina palveluna. Asiakkaan ei tarvitse asentaa tietokoneellensa itse mitään, koska ohjelma suoritetaan palveluntarjoajan tietokoneella eli palvelimella. SaaS-palveluja käytetään internet-selaimella tai esimerkiksi matkapuhelinsovelluksella, ja asiakkaan data tallennetaan palvelimella olevaan tietokantaan. SaaS-palvelut hinnoitellaan normaalisti käytön mukaan eikä kertamaksuna, kuten useat perinteiset ohjelmistot. Saatavilla on lukuisia SaaS-palveluita, kuten Salesforce[4] ja Facebook[5]. Lähes kaikki suuret ohjelmistoyritykset ovat alkaneet kehittää omia palveluitaan myös SaaS-mallin mukaisesti. [11]

Kaupallisesta näkökulmasta SaaS-palveluita tuotetaan eniten pienten ja keskisuurten yritysten tarpeisiin. Suurten yritysten tarpeet ovat yleensä niin vaativia, että niissä ohjelmistot valmistetaan edelleen alusta asti kyseisen yrityksen käyttöön. SaaS-palvelut voidaan jakaa tämän mukaan kahdenlaisiin toteutuksiin. Ensimmäinen kategoria sisältää palvelut pienten yritysten tarpeisiin. Niitä suunniteltaessa joudutaan optimoimaan valmistuskustannuksia mahdollisimman laajan asiakaskunnan tavoittamiseksi mahdollisimman edullisesti. Tällöin palvelut suunnitellaan niin, että sama palvelu vastaa mahdollisimman usean asiakkaan tarpeisiin mahdollisimman hyvin. Toinen kategoria on palvelut keskisuurilla yrityksillä. Keskisuurilla yrityksillä on usein oma IT-osastonsa ja enemmän tietotekniikkavalmiuksia, jolloin tarjottujen palveluiden vaatimukset voivat olla tarkemmat. Silloin palveluissa joudutaan usein tekemään räätälöityjä ominaisuuksia asiakastarpeen täyttämiseksi. [11]

[4] Salesforce on yrityksille tarkoitettu myynninedistämispalvelu. http://www.salesforce.com
[5] Facebook on maailman suurin yhteisöpalvelu. http://facebook.com

2.1.1 SaaS-palveluiden tuotekehitys

SaaS-palveluiden ohjelmistopäivitykset tehdään suoraan palveluntarjoajan tietokoneilla. Kaikki päivitykset tulevat kaikille asiakkaille automaattisesti, ja ohjelmisto voi muuttua paljonkin elinkaarensa aikana. Kun SaaS-palvelua aletaan suunnitella, ei voida vielä ennustaa mitä ominaisuuksia ja toiminnallisuuksia tuotteeseen tullaan tulevaisuudessa haluamaan. Perinteisessä ohjelmistomallissa ongelma on usein pienempi, koska ohjelmistosta voidaan julkaista aika ajoin uusi versio, jonka taaksepäin yhteensopivuutta ei aina tueta täydellisesti. SaaS-palveluiden kohdalla asiakkaat ovat tottuneet malliin, jossa palvelu kehittyy kaiken aikaa eikä uusia ohjelmistoversiota tarvita.

2.1.2 Tietoturva

SaaS-palvelut ovat kustannustehokkaita erityisesti pienille ja keskisuurille yrityksille, joilla ei muuten olisi välttämättä tarvittavia resursseja yhtä laajojen ohjelmistojen hankintaan ja ylläpitoon. SaaS-palveluita käytettäessä myös tietoturva on kokonaan palveluntarjoajan hallinnassa, eikä palvelua käyttävällä yrityksellä ole juurikaan mahdollisuuksia vaikuttaa palvelun käyttämään tekniikkaan tai palvelun tietoturvatasoon. Tästä syystä epävarmuus tietoturvatasosta on yksi suurimmista esteistä yrityksien siirtymisessä SaaS-palveluiden käyttöön. [12]

2.2 Pilvipalvelualustat

Pilvipalvelualusta eli IaaS (Infrastructure as a Service) on pilvipalveluiden alin kerros. Sen päälle rakennetaan ensin palvelin arkkitehtuuri, jonka päälle itse palvelu toteutetaan. Tunnetuimpia IaaS tarjoajia ovat Amazon ja Google. Pilvipalvelualustalla tarjotaan asiakkaille fyysisiä ja virtualisoituja palvelimia sekä hallintapalvelu, jonka avulla asiakas voi ”käynnistää” ja ”sammuttaa” itselleen palvelimia. [13]

IaaS:in käyttö tuo yritykselle monia etuja. Palvelinympäristön skaalaaminen isommaksi ja pienemmäksi on helppoa, kun palvelimia voi hankkia lisää ja vähentää ilman oikeita laitehankintoja ja asennuksia. Myös palvelimien käyttöaste saadaan suuremmaksi, kun useat yritykset käyttävät samoja palvelimia. Suuret pilvipalvelualustoja tarjoavat

yritykset pystyvät takaamaan palvelimilleen paremman toimintavarmuuden kuin pienet konesalit. Pilvipalvelimet voivat sijaita maantieteellisesti useilla eri mantereilla, ja tästä syystä niiden vikasietoisuus saadaan kasvatettua erittäin suureksi. [13]

Esimerkiksi Amazon EC2 palvelussa[6] asiakas maksaa palvelimien käytöstä vain käyttämältään ajalta. Tuntihintaan vaikuttaa esimerkiksi palvelimien tyyppi. Asiakas itse hallitsee vuokraamiensa palvelimien ohjelmistoja. Asiakasyrityksen tulee siis itse osata toteuttaa palvelinympäristö. Esimerkiksi Amazon tarjoaa kuitenkin joitakin ominaisuuksia valmiina toteutuksina, mm. kuormantasaajan (load balancer)[7] sekä virtuaalisen kiintolevyn[8]. Jos kuitenkin haluaa käyttää esimerkiksi relaatiotietokantaa, täytyy sen replikointi osata toteuttaa itse alustan päälle. Pilvipalvelualusta ei siis ole itsestään automaattisesti skaalautuva alusta palveluille vaan ennemminkin helppo ja yleensä halpa tapa järjestää yritykselle tarvittavat palvelimet.

2.3 Tietokanta

Tietokanta on järjestelmä, johon varastoidaan loogisesti yhteenkuuluvia asioita, kuten henkilörekistereitä, pankkitietoja tai juna-aikatauluja. Tietokannan avulla ohjelman logiikka ja tietosisältö (data) eriytetään toisistaan. Tietokantahallintajärjestelmän (Database Management System, DBMS) tehtävänä on tarjota valmiit rajapinnat fyysisen tietokannan hallintaan sekä tietokannan sisällön hakemiseen ja muokkaamiseen. [14]

Tietokannanhallintajärjestelmän tärkeimmät tehtävät:

1. **Tiedon fyysinen tallentaminen:** DBMS tallentaa tietokannan sisältämän tiedon useimmiten kiintolevylle käyttöjärjestelmän tarjoamassa tiedostoformaatissa.
2. **Eheyden varmistaminen:** Jos tietokannan sisältämä data pääsee korruptoitumaan, se voidaan menettää pysyvästi. Eheyden varmistamisen tehtävä on estää tiedon korruptoituminen.

[6] Amazon EC2 tarjoaa palvelimia pilvipalveluna. http://aws.amazon.com/ec2
[7] Amazonin kuormantasaaja. http://aws.amazon.com/elasticloadbalancing
[8] Amazonin virtuaalinen kovalevy S3. http://aws.amazon.com/s3

3. **Virheistä palautuminen:** DBMS yrittää säilyttää mahdollisimman suuren osan tietokannan sisällöstä laitevian (esimerkiksi kiintolevyn hajoaminen) sattuessa. Tietokantaa varmuuskopioidaan ja varmuuskopio voidaan palauttaa tarpeen tullen.
4. **Tietoturva:** Tietokantaa tulee voida käyttää vain oikeilla käyttäjäoikeuksilla.
5. **Datan käsittely:** Tietokannan sisällön muotoa tulee voida hallita ja dataa pitää pystyä lisäämään, muokkaamaan ja poistamaan.
6. **Ohjelmistorajapinta (API):** DBMS tarjoaa rajapinnan, jonka avulla voidaan ohjelmoida erillisiä tietokantaa käyttäviä järjestelmiä. Yleisin käytetty rajapinta on SQL-kielellä toteutettu liittymä. SQL-kielen avulla voidaan suorittaa kaikki tarvittavat toimenpiteet, mm. datan lisääminen ja poistaminen.

2.3.1 Tietokantataulut

Relaatiotietokannoissa tietokantojen sisältämän datan muoto esitetään tietokantatauluina. Tietokantataulu on kaksiulotteinen taulukko, jonka sarakkeet määrittelevät taulussa olevan datan muodon ja rivit sisältävät yksittäisiä tietoalkioita.

ID	etunimi	sukunimi
1	Matti	Meikäläinen
2	Ville	

Kuva 1. Esimerkki yksinkertaisesta henkilörekisteritietokantataulusta.

Tietokantatauluun määritellään sarakkeiden nimet ja niiden sisältämän tiedon muoto. Yksittäisen solun tieto voi olla muodoltaan esimerkiksi kokonaisluku tai määrämittainen merkkijono.

Tietokantatauluihin voidaan lisätä rivejä SQL-komentojen avulla. Yksinkertainen esimerkki henkilön lisäämisestä:

```
INSERT INTO kayttajat SET etunimi='Matti', sukunimi='Meikäläinen'
```

SQL-komentojen avulla voidaan lisätä, poistaa ja muokata olemassa olevia rivejä. Myös tietokantataulujen ja niiden sarakkeiden muokkaaminen onnistuu SQL-komennoilla.

2.3.2 Tietokantataulujen yhdistäminen kyselyissä

Relaatiotietokantojen yksi tehokkaimmista ominaisuuksista on taulujen yhdistäminen SQL-kyselyissä. Kyselyssä kahden tai useamman eri tietokantataulun rivejä voidaan yhdistää JOIN-komennoilla. Komennossa kerrotaan, mitä tauluja halutaan yhdistää ja millä säännöillä yhdistäminen tehdään. Näin voidaan kirjoittaa esimerkiksi kysely, joka hakee yhdestä tietokantataulusta pankin asiakkaan nimen ja yhdistää tulokseen kaikki asiakkaan tilit.

2.3.3 Indeksointi

Tietokannan suurin yksittäinen tehokkuuteen vaikuttava tekijä on tietokantataulujen indeksointi. Indeksoinnin idea on rakentaa tietokantamoottorille valmiita karttoja datan löytämiseksi. Esimerkiksi Kuva 1 esitetylle taululle voitaisiin tehdä indeksi sarakkeesta etunimi. Tällöin tietokanta tallentaa uuden indeksitaulun missä on vain sarake etunimi ja kopioi kaikki indeksoitavan taulun rivit aakkosjärjestyksessä indeksitauluun. Kun nyt tehdään kysely, jossa hakuehtona on tietty etunimi, ei tietokantamoottorin tarvitse käydä läpi tietokantataulun kaikkia rivejä vaan moottori voi hakea oikean kohdan suoraan järjestetystä indeksitaulusta.

Indeksien lisääminen lisää tietokannan käyttämän tilan tarvetta. Indeksit myös hidastavat tiedon kirjoittamista tietokantaan, koska tietokantamoottorin tulee aina päivittää myös tarvittavat indeksit. Siksi indeksien lisääminen pitää aina suunnitella ohjelmiston tarpeiden mukaisesti. Uutta ohjelmistoa suunnitellessa tietokannan sisältämän testidatan määrä on usein hyvin vähäistä. Tästä syystä usein ohjelmistoa kehittäessä tietokanta toimii lähes yhtä tehokkaasti, vaikka indeksointi jätettäisiin kokonaan tekemättä. Kun ohjelmiston tuotantokäyttö alkaa, tietokanta hidastuu datan lisääntyessä. Tässä vaiheessa indeksoinnin järkevä suunnittelu voi kuitenkin olla jo erittäin hankalaa.

Esimerkiksi Googlen kehittämä High Replication Datastore (HRD) NoSql-tietokanta tekee indeksin jokaisesta ohjelmiston tiedon hakemiseen käyttämästä sarakkeesta [15]. Tietokannasta ei siis haeta tietoa lainkaan tutkimalla läpi kaikkia tietokannan rivejä vaan

kaikki hakuehdot löytyy aina indekseistä. Etuna tässä mallissa on se, että indeksointi on aina järjestetty optimaalisesti. Vaikka HDR-tietokannassa datan muoto ei olekaan määritetty tarkasti sarakkeisiin, indeksit rakennetaan samankaltaisesti kuin relaatiotietokannoissakin. Lähes aina, kun relaatiotietokannasta tehdään hakuja, joita ei voida tehdä indeksien avulla, on kyseessä suunnitteluvirhe. Relaatiotietokannoissa hakujen tekeminen kuitenkin on mahdollista myös ilman indeksien käyttämistä. Tästä syystä järjestelmään voi jäädä epähuomiossa indeksoimattomia tietokantahakuja. Datan lisääntyessä ilman indeksejä suoritettavat haut tulevat joka tapauksessa hidastumaan.

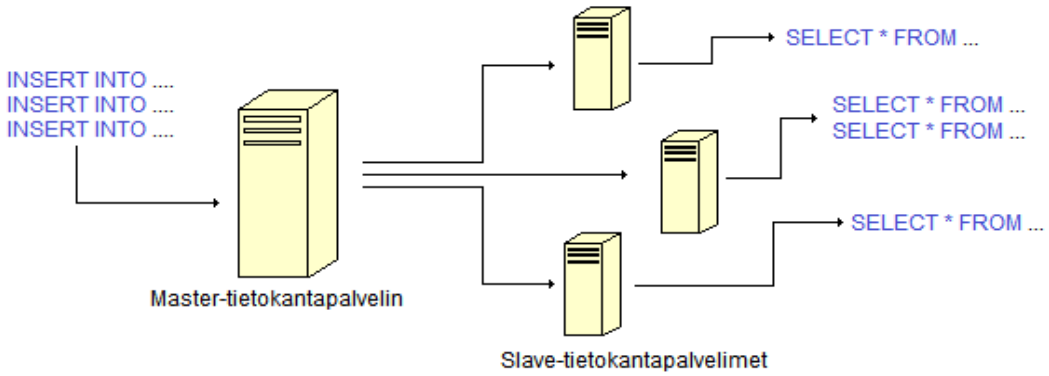
2.3.4 Skaalautuvuus

Skaalautuvuus tarkoittaa järjestelmän kykyä vastata kasvavaan käyttäjämäärään ilman, että järjestelmän arkkitehtuuria tarvitsee muuttaa [16]. Tietokannan skaalautuvuus voidaan toteuttaa joko lisäämällä tietokannan laskennan suorittavan palvelimen prosessointitehoa tai käyttämällä useampaa palvelinta klusterina. Prosessointitehoa voi lisätä vain rajoitetusti, ja tästä syystä raja skaalautuvuudessa tulee nopeasti vastaan. Klusterissa käytetään kahta tai useampaa tietokonetta rinnakkain laskentatehon lisäämiseksi. Tässä lähestymistavassa ongelmaksi muodostuvat relaatiotietokantaa käytettäessä tietokannan sisäiset suhteet. Relaatiotietokannassa kaiken datan tulee olla samalla fyysisellä tietokoneella, jotta tiedon haku voidaan suorittaa.

2.3.4.1 Relaatiotietokannan replikointi

Jos tietokantaa rasittaa enemmän tietokannassa suoritettavat haut kuin tiedon kirjoittaminen, voidaan relaatiotietokannan suorituskykyä kasvattaa niin sanotulla replikoinnilla [17][18]. Replikoinnissa yksi palvelin sisältää master-tietokannan, jota yksi tai useampi slave-tietokanta seuraa. Kun tietoa kirjoitetaan master-tietokantaan, slave-tietokannat kopioivat tiedot itsellensä pienellä viiveellä. Järjestelmän tulee kohdistaa kaikki tietokannan kirjoitusoperaatiot master-tietokantapalvelimelle. Tiedonhakukyselyt suoritetaan kuormantasausalgoritmin avulla tasaisesti kaikilla slave-tietokantapalvelimilla, kuten Kuva 2 on esitetty. Kuormantasausalgoritmeja on useita

erilaisia eri tarpeisiin, mutta yksinkertaisimmillaan ne voivat suorittaa satunnaisesti kyselyjä eri tietokantapalvelimilla. [19]



Kuva 2. Master-slave replikoinnin toimintaperiaate. Data kirjoitetaan master-palvelimelle, josta se replikoituu automaattisesti slave-palvelimille. Datan hakeminen suoritetaan slave-palvelimilta.

Master-tietokantaan kirjoitettu tieto kopioituu slave-tietokantaan aina viiveellä. Tämä viive pyritään saamaan mahdollisimman pieneksi (joitakin millisekunteja), mutta viive täytyy silti ottaa aina huomioon palvelun suunnitellussa. Yksinkertaisessa yhtä tietokantapalvelinta käyttävässä toteutuksessa voidaan aina olettaa, että tietokantaan kirjoitettu tieto löytyy heti seuraavassa kyselyssä. Kun käytetään master-slave-replikointia, saattaa palvelimien ruuhkautuessa viive kasvaa jopa useiden sekuntien mittaiseksi. Tällöin voi olla tarpeellista, että järjestelmä pitää erikseen omaa tilatietoa viimeisimmistä tietokantaan tehdyistä muutoksista.

2.3.5 Asiakkaiden datan hallinta

SaaS-palveluissa eri asiakkaiden data sijaitsee samassa tietokannassa. On tärkeää eristää tiedot niin, että niillä ei ole vaikutusta toisiinsa eivätkä asiakkaat näe toistensa tietoja. Eristämiseen on useita eri malleja. [20]

1) **Yksityiset taulut -mallissa** (Private Table Layout) jokaisella asiakkaalla on tietokannassa omat tietokantataulunsa. Tietokantataulut sijaitsevat joko yhteisessä tietokannassa tai jokaisen asiakkaan tietokantataulut ovat erillisissä tietokannoissa. Tällöin tietojen eristäminen toisistaan on helppoa suoraan tietokannan omilla ominaisuuksilla. Ongelmana on tietokannan ylläpitotyön lisääntyminen ja monimutkaistuminen. Jos asiakkuuksia on satoja tai tuhansia, on tietokantataulujen hallitseminen manuaalisesti liian raskasta. Tietokantojen tehokkuus myös laskee, jos tietokantataulujen lukumäärä kasvaa liian suureksi.

2) **Jaetut taulut -mallissa** (Shared Table Layout) kaikkien asiakkaiden data tallennetaan samoihin yhteisiin tietokantatauluihin. Jokaiseen tietokantariviin lisätään tieto siitä, kenelle asiakkaalle kyseinen rivi kuuluu. Tässä mallissa ongelmaksi tulevat erot asiakkaiden tarpeissa. Yhdelle asiakkaalle saatetaan lisätä tietokantaan sarake, jota kaikki asiakkaat eivät tarvitse. Tällöin täytyy kaikille niille asiakkaille, jotka eivät saraketta käytä, tallentaa kyseiseen sarakkeeseen NULL. Näin toimittaessa tietokannan sisällöstä voi suurin osa olla NULL tietoa.

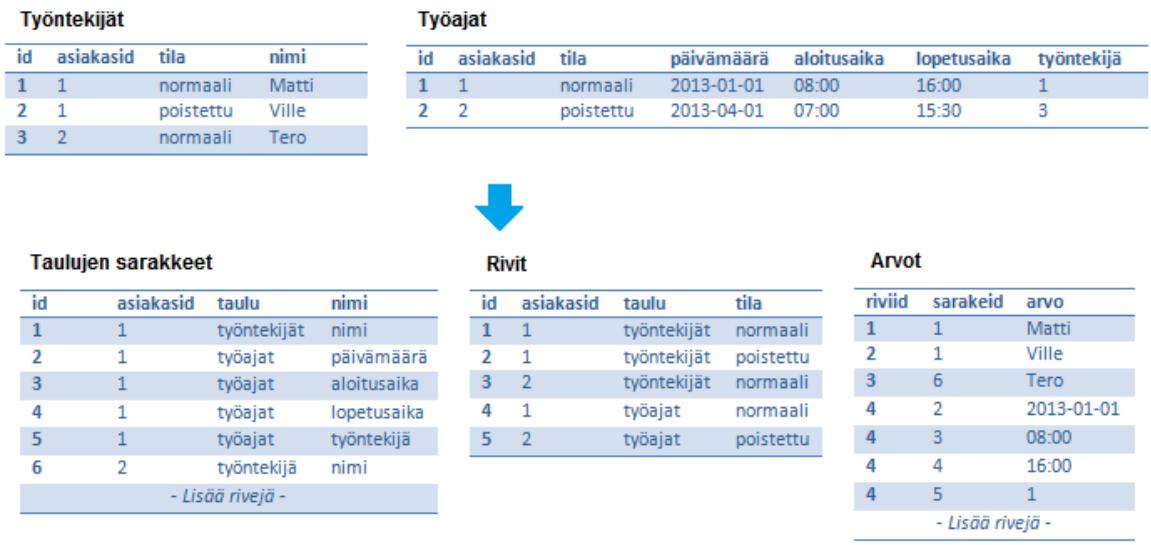
3) **Laajennustaulu-malli** (Extension Table Layout) yhdistää kahta aiempaa mallia. Siinä asiakkaille yhteiset sarakkeet lisätään yhteiseen tietokantatauluun ja asiakaskohtaiset lisäykset hoidetaan asiakaskohtaisilla laajennustauluilla. Yhteisen taulun ja laajennustaulun rivi tiedot sidotaan toisiinsa rivien tietokanta-avaimien perusteella. Mallilla on kuitenkin samat ongelmat kuin yksityiset taulut –mallilla, eli jos laajennustauluja tarvitaan paljon, niiden hallinta monimutkaistuu.

4) **Kokoojataulu-mallin** (Pivot Table Layout) ideana on jakaa rivit yksittäisiksi data-soluiksi. Yksittäiset solut tallennetaan kapeisiin tietokantatauluihin. Taulun jokaisella

rivillä on tieto, mihin riviin tieto liittyy, mikä sarake on kyseessä ja itse datan arvo. Tässä mallissa haasteena on tietomallin monimutkaisuus.

2.4 Movenium Collector

Movenium Collector on kauppanimi Movenium Oy:n itse kehittämälle tietokantaratkaisulle, jossa jokaiselle asiakkaalle luodaan omat virtuaaliset tietokantataulut. Collector on ohjelmoitu PHP (Hypertext Preprocessor) ohjelmointikielellä[9] ja sen tietokantana on MySql. Ratkaisu kehitettiin tarpeeseen räätälöidä jokaisen asiakkaan palvelua erikseen. Ideana on, että tietokannassa on ainoastaan jokaisen asiakkaan lomakkeiston määrittely sekä koontitauluja, joihin lomakkeiden data tallennetaan.



Kuva 3. Kuvassa on esitetty yksinkertaistettu malli Collectorin tietokantarakenteesta. Yllä normaali relaatiotietokantamuoto ja alla Collectorin rakenne.

Kuva 3 on esitetty, miten kaksi relaatiotietokannan taulua kuvataan Collectorin tietorakenteessa. Ensimmäinen taulu esittää yksinkertaista työntekijöitä sisältävää tietokantataulua ja toinen työaikoja sisältävää taulua. Näiden taulujen sarakkeet ovat

[9] PHP on suosittu skripti-kieli web-palveluiden toteutukseen. http://www.php.net/.

vain esimerkkejä. Alempana kuvassa on Collectorin tietokantataulut. ”Taulujen sarakkeet”-taulu kuvaa kaikki järjestelmään määritellyt virtuaaliset taulut ja niiden sarakkeet. ”Rivit”-taulussa on yksi rivi jokaisesta työntekijä- ja työaikarivistä, jotka tietokantaan on lisätty. ”Arvot”-taulu sisältää kaikki tietokannan varsinaiset arvot. Arvot on indeksoitu rivin ja sarakkeen avainten mukaan. Näin Collectorin tietokantataulujen muoto ei muutu vaikka järjestelmän virtuaalitaulujen muoto muuttuisikin.

Tietorakenteen lisäksi Collector tarjoaa valmiit funktiot tiedon kirjoittamiselle ja hakemiselle tietokannasta. Tiedon kirjoittamiseen on funktio, jolle annetaan parametreina taulun nimi, johon tieto halutaan kirjoittaa sekä taulukko, joka sisältää rivin sarakkeisiin kirjoitettavan datan. Hakufunktiolle annetaan taulun nimi, josta tietoa haetaan sekä taulukko, joka sisältää hakuehtoja tiedon suodattamista varten. Funktioiden olemassaolo on tärkeää, koska monimutkaisen tietokantarakenteen takia SQL-muotoisten hakujen kirjoittaminen on hyvin haastavaa. Koontitaulujen tietojen yhdistäminen kyselyissä hoidetaan JOIN-komennoilla. Collectorin hakufunktiota kutsuttaessa annetaan parametreina, mistä virtuaalitaulusta tietoja haetaan ja millä hakuehdoilla. Näiden perusteella Collector rakentaa SQL-kyselyn, suorittaa sen ja palauttaa haetun datan taulukkomuodossa.

3 Tietokantaratkaisut ja kehitysmenetelmät kirjallisuudessa

Tässä kappaleessa tarkastellaan, mitä vaihtoehtoja tietokantaratkaisuksi on olemassa ja miten hyvin ne täyttäisivät Movenium Oy:n palveluiden tarpeet.

Tietokantaratkaisun valinta on tärkeä päätös, kun aletaan kehittämään uutta ohjelmistoa. Tietokannan vaihtaminen jälkikäteen voi olla työnä moninkertainen alkuperäiseen toteutukseen verrattuna. Valintaan vaikuttavat esimerkiksi kehittäjien aiempi kokemus, tietokannan käytön aloittamisen helppous ja monet muut tekijät. Valinnassa olisi kuitenkin tärkeintä analysoida, kuinka hyvin tietokantaratkaisu palvelee ohjelmiston tarpeita jatkossa, kun skaalautuvuus ja tehokkuustarpeet kasvavat. Tämä pitää erityisesti paikkaansa SaaS-palveluiden kohdalla, joissa yksi tietokanta palvelee jopa miljoonia käyttäjiä. Analyysin tekeminen voi kuitenkin olla erittäin vaikeaa. Aloittavilta yrityksiltä puuttuu tässä vaiheessa usein resurssit asian tutkimiseen eikä palvelun elinkaarta vielä välttämättä osata ennustaa.

3.1 Relaatiotietokannat

Relaatiotietokannat kehitettiin jo 1970-luvulla ja ne ovat edelleen yleisin tietokantamalli erilaisten palveluiden taustalla. IBM kehitti relaatiotietokantojen käyttämiseen SQL-kielen, jonka avulla tietokannan rakennetta hallitaan ja tietokannan sisältöä sekä haetaan että lisätään. SQL-kieli tarjoaa syntaksin tietokantakomentojen rakentamiseen. Nämä komennot toimivat rajapintana tietokannan ja ohjelmointiin käytettävän ohjelmointikielen välillä. [21]

Relaatiotietokantamalli on erityisen sopiva yksinkertaiseen työaikojen tallentamiseen. Työajat tallennetaan tietokantaan yksittäisinä riveinä ja SQL-kielen avulla voidaan suoraan rakentaa hakuja erilaisia raportointitarpeita varten. Työaikojen kohdistaminen projekteihin ja työntekijöihin hoidetaan tietokantataulujen välisillä suhteilla, kuten Kuva 4 esitetty. Kuva esittää kolmea relaatiotietokannan taulua. Työajat-taulussa on yksi rivi jokaista työaikaa kohden, joista jokaisessa on viittaus sekä työntekijätauluun (työntekijäid sarake) että projektitauluun (projektiid sarake). Viittausten arvona on viitatun taulun rivin avain. Näiden viittausten mukaan SQL-kielen avulla voidaan

kirjoittaa kysely, joka palauttaa työajat suoraan siten, että ne sisältävät työntekijöiden ja projektien nimet.

Kuva 4. Työaikojen yksinkertaistettu malli relaatiotietokannassa.

Relaatiotietokannan rakennetta kutsutaan datamalliksi. Datamalli on tietokannan rakenteen fyysinen esitys kaikista tietokannan komponenteista ja niiden välisistä suhteista. Datamalli määrittää vähintään seuraavat asiat: [21]

- Tietokannan sisältämät taulut
- Sarakkeet joista taulut koostuvat
- Sarakkeiden tyypit
- Avaimen eli tiedon siitä missä tietokantataulun sarakkeessa on rivin yksilöivä tieto
- Taulujen väliset suhteet, eli esimerkiksi viittaukset yhden taulun solusta, toisen taulun riviin

Uutta sovellusta suunnitellessa kaikki edellä kuvatut asiat tulee siis päättää valmiiksi jo ennen sovelluksen ohjelmoimista. Datamallia voidaan muuttaa vielä ohjelmoinnin aikana, mutta datamallin muuttaminen vaatii usein myös jo ohjelmoidun osuuden muuttamista vastaamaan uutta datamallia. Työaikojen datamallin suunnittelussa tämä osoittautuu ongelmaksi. Jos kehitetään datamalli esimerkiksi pankkitilitietojen

tallentamiseen, voidaan datamalli suunnitella valmiiksi asti ennen ohjelmoinnin aloittamista ja sama malli toimii kaikille asiakkaille. Työaikojen datamalli poikkeaa tässä suuresti, koska työaika voi sisältää hyvinkin erilaista dataa eri yrityksien välillä.

Yleisesti käytettyjä relaatiotietokantoja ovat muun muassa MySQL[10], PostgreSQL[11] sekä Oracle Database[12]. Näistä MySQL ja PostgreSQL ovat avoimeen lähdekoodiin perustuvia ja Oracle Database on maksullinen järjestelmä. Jokainen näistä tietokannoista toimii SQL-kielen avulla. SQL-kieli on kaikissa näissä lähes identtistä, mutta pieniä eroja löytyy. Esimerkkinä MySQL-tietokannassa merkkijonojen vertailussa isot ja pienet kirjaimet käsitellään samoina, mutta PostgreSQL pitää isoja ja pieniä kirjaimia eri merkkeinä. Näiden pienien erojen takia tietokantaa ei voida ilman SQL-kyselyiden läpi käymistä vaihtaa toiseen. Tähän ongelmaan on kuitenkin ratkaisuja, joissa tietokanta abstraktoidaan kirjaston avulla. Tällainen kirjasto on esimerkiksi PHP-kielessä PDO[13].

MySQL on suosituin tietokanta web-sovelluksissa [22]. MySQL perustuu avoimeen lähdekoodiin ja on näin ollen käyttäjälle ilmainen. Tämä tekee käytön aloittamisesta helppoa. Ensimmäinen versio on julkaistu jo vuonna 1994, joka tekee MySQL:stä yhden vanhimmista avoimen lähdekoodin tietokannoista. MySQL sisältää kaikki relaatiotietokannan tärkeimmät ominaisuudet, mutta ei esimerkiksi tue täydellisestä SQL-standardin mukaista viiteavain[14] (foreign key) tekniikkaa [23]. Viiteavaimilla viitataan tietokannassa yhdestä tietokantataulusta toiseen. MySQL tietokannan hallinnoimiseen löytyy useita ilmaisia ja maksullisia ohjelmia. Näistä esimerkkeinä phpMyAdmin[15] sekä Windows sovellus SQLyog[16]. Movenium Oy käyttää tällä hetkellä kaikissa tuotteissaan alustana MySQL-tietokantaa.

---

[10] MySQL on avoimeen lähdekoodiin perustuva relaatiotietokanta. http://www.mysql.com/.
[11] PostgreSQL on avoimeen lähdekoodiin perustuva relaatiotietokanta. http://www.postgresql.org/.
[12] Oracle Database on oraclen kehittämä tietokanta. http://www.oracle.com/us/products/database/overview/index.html
[13] PDO on tietokannan abstraktointi kirjasto. http://fi2.php.net/manual/en/intro.pdo.php.
[14] Viiteavaimilla tietokannassa viitataan toisiin tauluihin. http://en.wikipedia.org/wiki/Foreign_key.
[15] phpMyAdmin on PHP-kielellä toteutettu avoimen lähdekoodin MySQL hallintatyökalu. http://www.phpmyadmin.net/home_page/index.php.
[16] SQLyog on Windows ympäristössä toimiva graaffinen MySQL hallintatyökalu. https://www.webyog.com/.

Myös PostgreSQL on avoimeen lähdekoodiin perustuva tietokanta. Yksi merkittävä ero MySQL:ään verrattuna on PostgreSQL:ssä käytetty MVCC-tekniikka (multiversion concurrency control) [24]. Tekniikassa tietokannassa suoritettavalle transaktiolle annetaan ”kuva” tietokannan sen hetkisestä tilanteesta, jota transaktio voi muokata. Näin samaan aikaan suoritettavat transaktiot voidaan suorittaa toisistaan riippumattomasti [25]. MySQL-tietokannassa sama asia hoidetaan tietokantataulujen lukituksilla, mikä hidastaa tiedon kirjoittamista tietokantaan.

Oracle Database on tässä esitellyistä tietokannoista ainoa maksullinen järjestelmä. Oracle Database tietokantaa käyttävät usein isot yritykset ja valtionlaitokset, koska Oracle tarjoaa tuotteilleen maksullista tukea[17]. Avoimen lähdekoodiin perustuvia järjestelmiä käytettäessä yritys ei voi ongelmatilanteessa tukeutua järjestelmän toimittajaan.

3.2 NoSql

NoSql [26] ei tarkoita yhtä tiettyä teknologiaa. NoSql-tietokannaksi voidaan kutsua käytännössä kaikkia tietokantaratkaisuja, jotka voivat sisältää suuria määriä dataa (useita teratavuja), mutta jotka eivät perustu perinteiseen relaatiomalliin. NoSql-ratkaisut ovat melko uusia tekniikoita. Ensimmäiset toteutukset ovat noin 2000-luvun puolivälistä [27]. NoSql-tietokannat eivät ole vielä yhtä laajasti käytössä kuin relaatiotietokannat, mutta niiden suosio kasvaa nopeasti[18]. NoSql on tullut nopeasti tunnetuksi, koska esimerkiksi Google on kehittänyt itselleen suuren skaalautuvan infrastruktuurin sen pohjalle.

3.2.1 Käytetyimmät NoSql-tietokannat

Nykyään on olemassa yli 200 julkaistua NoSql-ratkaisua[19]. Näistä kolme tunnetuinta ovat Big Table, MongoDB ja Cassandra.

---

[17] My Oracle Support. https://support.oracle.com/.
[18] RDBMS dominate the database market, but NoSQL systems are catching up. DB-Engines.com. http://db-engines.com/en/blog_post/23. Viitattu: 22.2.2014.
[19] Lista NoSql-tietokannoista. http://en.wikipedia.org/wiki/NoSQL. Viitattu: 22.2.2014.

Googlen kehittämä Big Table oli ensimmäisiä NoSql-tietokantoja kun se julkaistiin vuonna 2004. Google toteutti Big Table-tietokannan päälle omia palveluitaan, kuten Google Earth[20] ja Google Analytics[21]. Tietokannan datamalli on avain-arvo (key-value) tyyppinen. Malli perustuu järjestettyyn karttaan, joka on indeksoitu kolmen arvon mukaan: rivin avaimen, sarakkeen avaimen ja aikaleiman. Tietokannassa varsinainen data on esitetty merkkijonona, joka voi sisältää esimerkiksi JSON-muotoista dataa. Jokaista arvoa varten on siis oma avaimensa ja koko tietokanta on järjestetty näiden avaimien perusteella. [28]

MongoDB on C++ kielellä toteutettu avoimeen lähdekoodiin perustuva projekti, jonka tavoitteena on tehdä hyvin tehokas tietokantaratkaisu. MongoDB tukee horisontaalista skaalausta sekä master-slave -replikointia, jota hyödynnetään virhe tilanteista palautumiseen ja varmuuskopiointiin. Olemassa olevan tietokantarakenteen yhteensovittaminen NoSql-tietokannan kanssa voi olla iso haaste toimijoille, joiden palvelut toimivat jo relaatiotietokannan päällä. [29]

Myös Cassandra mahdollistaa horisontaalisen skaalauksen tietokoneklusterilla. Cassandran kehitti alkuun Facebook ja vuonna 2008 se julkaisi lähdekoodin avoimeksi (Facebook kuitenkin itse lopetti Cassandran käytön vuonna 2010). Tietokantaan lisätty tieto leviää tietokannassa paikoilleen tietyn ajan sisällä ja tämän ajan jälkeen tieto löytyy jokaisella kyselyllä. [30]

3.2.2 NoSql-tietokantojen suorituskyky

Relaatiotietokantaan verrattuna NoSql on helpommin skaalattava, koska NoSql tukee horisontaalista skaalausta. Vertikaaliseksi skaalaukseksi sanotaan mallia, missä tietokannan tehoa lisätään käyttämällä mahdollisimman tehokasta usean ytimen tietokonetta. Horisontaalinen malli perustuu usean tietokoneen käyttämiseen rinnakkain klusterina. Yksittäisen tietokoneen teholla on aina fyysiset rajat ja hyvin tehokkaat tietokoneet ovat kalliita, mutta klusteriin voidaan aina liittää lisää

[20] Googlen karttapalvelu. http://www.google.com/earth/.
[21] Googlen verkkoanalysointityökalu. http://www.google.fi/analytics/.

tietokoneita. Suurimmissa NoSql-tietokannoissa voi olla rinnakkain tuhansia tai jopa satoja tuhansia tietokoneita. [31]

Relaatiomallissa tietokannan kaikki data tulee olla yhdellä tietokoneella. Tämä johtuu relaatiomallin tavasta yhdistää dataa useasta eri taulusta JOIN-komentoja käytettäessä. Relaatiotietokantaa voidaan skaalata myös horisontaalisesti replikoinnin avulla. Rajoitteena on kuitenkin se, että yksittäinen tietokantakysely suoritetaan relaatiotietokannassa aina alusta loppuun yksittäisellä tietokoneella. Eli vaikka replikoituja tietokantoja olisikin useita, se ei nopeuta yksittäisen kyselyn suorittamista. Tämän takia tietokannan koon kasvaessa, jokaisen skaalaukseen käytetyn tietokoneen tulee olla tarpeeksi tehokas suorittamaan kysely, koko tietokannan sisältämästä data määrästä.

NoSql-tietokantojen kanssa tilanne on toinen. Skaalautuvuus voidaan ratkaista käyttämällä horisontaalista skaalausta ja datan hajauttamista. NoSql-tietokannan sisältämä data voidaan jakaa useammalle tietokoneelle. Kyselyn suorittaminen voidaan hajauttaa rinnakkain monelle eri tietokoneelle. Rinnakkaissuorittaminen on mahdollista MapReduce-tekniikan ansiosta [31]. Tekniikan ideana on suorittaa kysely kahtena peräkkäisenä prosessina. Ensimmäinen vaihe on kartoitus (map). Kartoitus suoritetaan rinnakkain kaikilla tietokannan tietokoneilla. Jokainen tietokone käy läpi oman datansa ja palauttaa kaikki hakuun sopivat tulokset. Toinen vaihe on sievennys (reduce). Sievennyksen tarkoitus on vastaanottaa kaikki kartoituksen tuloksena saadut tulokset ja sieventää tulos täyttämään haun vaatimukset.

Tärkeimmät NoSql:n edut perinteiseen relaatiotietokantaan ovat 1) kirjoitus- ja lukuoperaatioiden nopeus, 2) tuki erittäin suurille datamäärille (useita teratavuja), 3) helppo tietokannan laajentaminen ja 4) halpa hinta tehokkuuteen nähden [32][33]. Toisaalta NoSql-mallit eivät yleensä sisällä valmista tukea SQL-kielelle eikä transaktioille. Varsinkin SQL-kielen puute voi olla esteenä NoSql:n käyttöönotossa. SQL-kieli on teollisuusstandardi, jonka suuri osa ohjelmistoalan ihmisistä hallitsee. SQL-kielen puute tarkoittaa myös, että NoSql-järjestelmissä raporttien luominen tietokannan sisältämästä datasta ei onnistu yhtä suoraviivaisesti kuin relaatiotietokantaa

käytettäessä. Ohjelmoijan täytyy tuntea käytössä olevan tietokannan sisäinen rakenne ja logiikka paljon paremmin kuin SQL-kieltä käytettäessä.

NoSql-tietokantoja kuitenkin kehitetään voimakkaasti, koska relaatiotietokantamalli ei ole tarpeeksi tehokas monille järjestelmille. NoSql on myös hyvä vaihtoehto järjestelmiin, joissa tiedon oikeellisuus ja viiveettömyys eivät ole välttämättömiä. Tällaisesta palvelusta on hyvänä esimerkkinä erilaiset blogi-palvelut. Niissä ei ole haitallista, jos esimerkiksi jokin uusi kirjoitus näkyisikin joillekin käyttäjille muutaman sekunnin viiveellä.

3.2.3 NoSql ja tietoturva

MongoDB:n käyttämä JavaScript-pohjainen sisäinen komentokieli on haavoittuvainen injektiohyökkäyksille [34], jollei ohjelmoija muista tai osaa ottaa asiaa huomioon. Injektiohyökkäyksessä palvelun loppukäyttäjä antaa tietokannalle omia komentojaan syöttämällä niitä palvelun kenttiin (esimerkiksi salasanakenttä), jotka tietokanta sitten suorittaa. Klassinen esimerkki injektiohyökkäksestä on kirjoittaa käyttäjänimeksi esimerkiksi "'; DROP TABLE users; --", jolloin DROP TABLE käsky suoritetaan tietokannassa[22].

MongoDB ja Cassandra eivät sisällä valmiita ominaisuuksia sisältönsä suojaamiseen salauksen avulla. Myös molempien tietokantojen käyttämät autentikointimetodit ovat melko heikkoja [34]. Cassandran autentikointi tapahtuu lähettämällä tietokannalle käyttäjätunnus ja salasana. Molemmat lähetään suojaamattomana. Salasanalle voidaan erikseen asettaa, että salasanan täytyy olla MD5 [35] muodossa. MD5 suojaus on kuitenkin murrettavissa rainbow-taulukoiden[23] avulla [36]. Jos hyökkääjä siis onnistuu pääsemään kiinni tietokantaa käyttävän järjestelmän ja tietokannan väliin, voi hyökkääjä onnistua saamaan tunnukset itsellensä. MongoDB suojaa salasanat myös MD5-algoritmilla, mutta turvallisuutta lisää salasanan suolaus (salting) [37]. Suolauksessa salasanaan liitetään ylimääräisiä merkkejä ennen MD5-algoritmin käyttöä. MongoDB:n

[22] SQL-injektio wikipediassa: http://fi.wikipedia.org/wiki/SQL-injektio. Viitattu: 6.3.2014.
[23] Esimerkki rainbow-tauluista internetissä. http://project-rainbowcrack.com/table.htm.

tapauksessa MD5-algoritmilla suojataan merkkijono <username>:mongo:<password>. Näin tehden salasanaa ei voida palauttaa, mutta mahdolliseksi jää edelleen brute-force-tekniikan[24] käyttäminen.

3.2.4 NoSql:n käyttäminen työajanseuranta palvelussa

Työaikojen tallentamiseen sopiva malli olisi niin sanottu Column Oriented Database[25]. Mallissa tietoa ei tallenneta tietokantaan tarkkaan määriteltyihin tauluihin, vaan joka rivillä on yksi vaihtuvamittainen sarake. Tähän sarakkeeseen voidaan tallentaa mitä tahansa tietoja sarjallistettuna esimerkiksi JSON-muodossa esimerkiksi näin:

{23456: {date: ”2013-08-22”, starttime: ”12:00”, endtime: ”13:15”, employeeID: ”573”}}

Tällä tekniikalla asiakkaiden eri tarpeet työajan muodolle voitaisiin ratkaista siten, että työaikojen sallittaisiin olla erimuotoisia.

3.3 Tietokanta palveluna (DBaaS)

Tietokannan ylläpito vaatii yritykseltä aina resursseja. Kun tiedon määrä tietokannassa kasvaa ja käyttäjiä tulee lisää, on tietokannan ylläpito aina vaativampaa. Siihen kuluu enemmän työaikaa ja siihen tarvitaan enemmän asiantuntemusta. Useampi suuri internet-alan yritys (Yahoo!, Google, Amazon) on kehittänyt tähän tarpeeseen tietokantaratkaisuja, joissa tietokantaa ei ylläpidetä itse vaan se hankitaan palveluna. [38]

Palveluna tarjottavat tietokannat ovat tällä hetkellä vielä melko uusi tekniikan ala. Googlen uusi BigTable-tekniikkaan perustava tietokanta on ensimmäinen suuren mittakaavan tietokantapalvelu, joka tukee ACID-transaktioita [39]. ACID-transaktiot takaavat, että tieto on tietokannassa kaikissa tilanteissa ehyttä. Relaatiotietokannan sisältämän datan tulee aina täyttää ACID-vaatimus [40]. DBaaS-palveluiden käyttäminen

[24] Brute-force-tekniikassa kokeillaan järjestyksessä kaikkia mahdollisia vaihtoehtoja suurella nopeudella. http://en.wikipedia.org/wiki/Brute-force_attack.

[25] Column Oriented Database –tekniikassa tiedot tallennetaan pitkiksi merkkijonoiksi. http://en.wikipedia.org/wiki/Column-oriented_DBMS. Viitattu: 6.3.2014.

voisi vähentää yritysten oman tietokanta-asiantuntemuksen tarvetta huomattavasti, jos sama tietokantaratkaisu olisi alussa halpa ja datamäärän kasvaessa automaattisesti skaalautuva.

3.4 Ruby on Rails

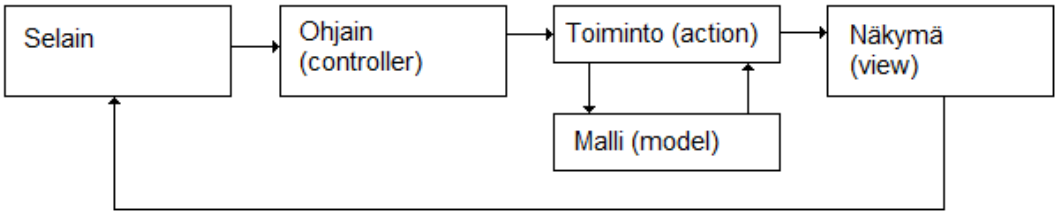
Ruby on Rails [41][42] ei ole tietokantaratkaisu vaan olio-ohjelmointikieli, joka on suunniteltu tehokkaaseen ja nopeaan internet-palveluiden kehittämiseen. Ruby on Rails:in kanssa käytetään usein MySql tietokantaa. Rubyllä palvelun datamalli määritellään kertaalleen Ruby on Rails -kielen avulla ja Ruby tuottaa sen avulla tarvittavat ohjaustiedostot. Samalla datamallin määrittelyllä Ruby hoitaa itse tietokannan muokkaamisen sekä tuottaa valmiin HTML-muotoisen taulukon, joka näyttää tietokannan yksittäisen taulun sisällön. Ruby luo myös valmiin HTML-lomakepohjan, jolla tietoa voidaan lisätä tietokantaan. [43]

MVC (Model View Controller) on ohjelmoinnissa usein käytetty malli[26] ja Ruby on Rails käyttää aina MVC-mallia. Mallia voi käyttää ohjelmoidessaan lähes millä ohjelmointikielellä tahansa, mutta Ruby on Rails pakottaa sen käyttämiseen. Perinteinen tapa ohjelmoida internet-sovellus on ollut kirjoittaa kokonainen internet-sivu yhteen tiedostoon, joka vastaanottaa kutsun, tekee tarvittavat toimenpiteet ja renderöi sivun. Nykyiset internet-palvelut ovat kuitenkin niin laajoja, että toimintoja täytyy ohjelman selkeyden säilyttämiseksi jakaa eri osiin.

Kuva 5 on esitetty MVC-mallin toiminta. Siinä ohjain (controller) vastaanottaa pyynnön selaimelta ja suorittaa tarvittavat toiminnot (action). Tämän jälkeen ohjain antaa tuloksensa näkymälle (view), jonka tehtävänä on renderöidä selaimelle palautettava sivu ohjaimen antaman tiedon perusteella. Malli (model) sisältää palvelun varsinaisen logiikan ja tietokantapalvelut. MVC-mallin tarkoitus on siis jakaa ohjelman toiminnot

[26] Yksinkertainen esimerkki MVC-mallin käytöstä: http://www.codeproject.com/Articles/25057/Simple-Example-of-MVC-Model-View-Controller-Design. Viitattu: 6.3.2014.

omiin lokeroihinsa, jolloin ohjelmakoodin ymmärrettävyys ja kehitettävyys pysyy hyvänä. [44]



Kuva 5. Kaaviokuva MVC-mallista.

Collector itsessään voidaan ajatella MVC-tekniikan termein mallina (model). Collectorin avulla voidaan siis luoda erilaisia malleja ilman ohjelmointia tai kehittämistä. Collectorin tehtäviin kuuluu tietokannan hallinta sekä palveluiden taustalla olevan logiikan laskeminen. Taustalogiikkaa voidaan määritellä asetuksilla. Asetuksen avulla voidaan esimerkiksi säätää, että aina kirjoitettaessa rivi tauluun A, kutsutaan funktiota B. Tällä tavoin voidaan toteuttaa mahdollisimman vähällä työllä logiikkaa mallin taakse. Tapa mahdollistaa myös asiakaskohtaisten lisätoimintojen tekemisen ilman, että muiden asiakkaiden palvelut häiriintyvät.

Ruby on Rails on tehokas tekniikka SaaS-palveluiden toteuttamiseen. Tekniikka ei kuitenkaan tarjoa valmiita vastauksia dynaamisen, asiakaskohtaisesti säädettävän, palvelun toteuttamiseen. Ruby on Rails on parhaimmillaan, kun toteutetaan pienten yritysten käyttöön soveltuvia palveluita, joissa kaikkien asiakkaiden palvelu toimii samalla tavalla.

3.5 Päätelaitesovellukset

Yksi tapa säästää palvelimien kuormaa on siirtää ohjelman suorittamisesta osa päätelaitteille [45]. Mobiililaitteiden suosio kasvaa kaiken aikaa PC-tietokoneisiin verrattuna ja suosion odotetaan kasvavan edelleen [46]. Mobiililaitteiden suorituskyky on niin hyvä, että SaaS-palveluissa voidaan iso osa ohjelman suorittamisesta siirtää päätelaitteen tehtäväksi.

Selain-pohjaisia palveluiden suorittaminen päätelaitteella voidaan toteuttaa esimerkiksi Javascript-ohjelmointikielen[27] avulla. Internet-selaimet ovat tukeneet Javascript-kieltä jo 2000-luvun alusta asti, mutta eri selaimet käsittelivät koodia hieman eri tavoilla [47]. Tästä johtuen Javascript-kielen suosio on lisääntynyt vasta yhteensopivuusongelmia poistavien kirjastojen myötä. Yksi esimerkki tällaisesta kirjastosta on jQuery[28]. Kirjaston avulla monet toimenpiteet, jotka pelkkää Javascript-kieltä käyttämällä ovat monimutkaisia toteuttaa, hoituu jQuery:n tarjoamilla funktiolla helposti. Esimerkkinä mainittakoon HTML sivun DOM-rakenteen hallinta[29].

Javascript-ohjelmointikielen avulla toteutetaan suuria kokonaisuuksia. Ohjelmoinnin tehostamiseksi on kehitetty ohjelmointialustoja (Framework) helpottamaan Javascript-pohjaisten sovellusten tekemistä. Esimerkkejä tällaisista alustoista ovat AngularJS[30] sekä Ember.js[31].

AngularJS on googlen ylläpitämä avoimeen lähdekoodiin perustuva ohjelmointialusta. Alustan avulla voidaan toteuttaa MVC-mallin mukaisia yhden sivun selainsovelluksia. Tämä tarkoittaa sitä, että kun sovelluksen loppukäyttäjä menee selaimellaan sovelluksen internet-osoitteeseen, latautuu palvelu kokonaan käyttäjän päätelaitteelle. Tästä eteenpäin internet-yhteyden yli ollaan taustapalvelimeen yhteydessä vain kun se on välttämätöntä. Keskustelu taustapalvelimen kanssa voidaan toteuttaa esimerkiksi JSON-muotoa käyttävän RESTful[32] ohjelmistorajapinnan ylitse [48]. Näin tietoliikenneyhteyden käyttöaste saadaan laskettua alhaiseksi.

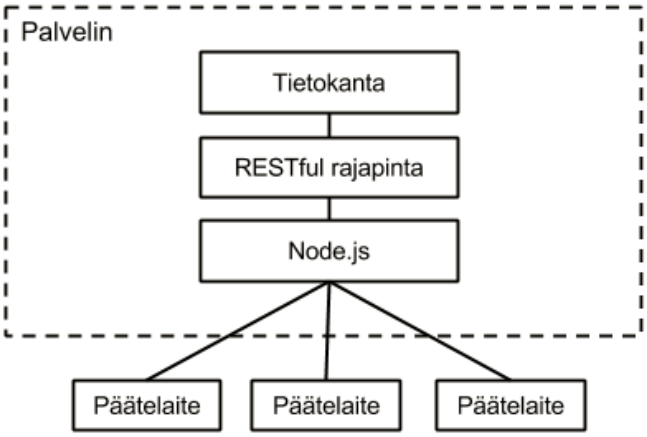
Ember.js on myös avoimeen lähdekoodiin perustava ohjelmointialusta ja myös sen avulla rakennetaan yhden sivun internet sovelluksia. AngularJS ja Ember.js ovat siis käytännössä kilpailevia tekniikoita. Molemmat ovat vielä niin uusia, että valinta näiden

---

[27] Javascript on internet selaimessa suoritettava skriptikieli. https://developer.mozilla.org/en-US/docs/Web/JavaScript.
[28] jQuery on ohjelmointia helpottava Javascript-kirjasto. http://jquery.com/.
[29] Dokumentaatio jQuery:n DOM-rakenteen läpikäymisfunktioille. https://api.jquery.com/category/traversing/tree-traversal/.
[30] AngularJS on Javascript ohjelmointialusta. http://docs.angularjs.org/guide/introduction.
[31] Ember.js on Javascript ohjelmointialusta. http://emberjs.com/.
[32] RESTful API on HTTP:n ominaisuuksia hyödyntävä ohjelmistorajapinta.

välillä on melko vaikeaa ja molemmilla on omat kannattajansa. AngularJS:n ensimmäinen vakaa versio on julkaistu vuonna 2012[33] ja ember.js:sän vuonna 2011[34]. Eräs ember.js-alustan pääkehittäjistä, Yehuda Katz[35], on ollut kehittämässä myös muun muassa Ruby on Rails:ia sekä jQuery:ä.

Päätelaitesovelluksen ja taustapalvelimen välissä voidaan käyttää vielä yhtä tehokkuutta lisäävää tekniikka. Node.js[36] ohjelman avulla taustapalvelimelle voidaan kirjoittaa Javascript-kieltä käyttäen esimerkiksi lisää rajapintakomentoja [49]. Näin toimiessa päätelaitteella toimivan sovelluksen ja taustapalvelimen välillä tarvittavaa liikennettä voidaan edelleen vähentää.



Kuva 6. Node.js ohjelman käyttö palvelimella.

Kuva 6 nähdään kuinka node.js ohjelma asemoituu palvelimen muiden ohjelmistojen joukkoon. Ääriviivoilla rajattu suorakulmio esittää mitkä kuvan toiminnoista tapahtuu palvelimella. Päätelaitteet siis keskustelevat node.js ohjelman kanssa, joka taas käyttää RESTful-rajapintaa tietokannan kanssa keskustelemiseen.

[33] AngularJS versioden 1.0 ja 1.2 roadmap. http://blog.angularjs.org/2012/07/angularjs-10-12-roadmap.html. Viitattu: 28.2.2014.
[34] Ember.js julkaisu uutinen. http://www.h-online.com/open/news/item/SproutCore-2-0-becomes-Ember-js-1394362.html. Viitattu: 28.2.2014.
[35] Yehuda Katzin kotisivu. http://yehudakatz.com/.
[36] Node.js on alusta internet sovellusten toteuttamiseen. http://nodejs.org/.

Päätelaitesovellusten ja palvelimella toimivien ohjelmien kehittäminen on hyvin erilaista. Päätelaitteille ohjelmoitaessa joudutaan kiinnittämään huomiota käyttöliittymien toimivuuteen kun taas palvelimien tietokantaa käsittelevien ohjelmien toteutuksessa tärkeää on tietoturva ja kuormankestävyys. Myös ohjelmointikieli on eri. Node.js-ohjelman käyttäminen palvelimella mahdollistaa, että päätelaiteohjelmointiin erikoistuneet ohjelmoijat, jotka eivät hallitse palvelinohjelmointia, pystyvät itse ohjelmoimaan taustapalvelimen rajapintaan uusia toiminnallisuuksia [50]. Päätelaiteohjelmoija voi toteuttaa uusia toiminnallisuuksia käyttäen itselleen tuttua Javascript-kieltä.

4 Collectorin kuormankestävyyden tutkiminen

Movenium Oy:n käyttämän virtuaalisen tietokannan eli Collectorin tämän hetken yksi tärkeimmistä kehityskohteista on tietokantaratkaisun kuormankestävyys kun palveluiden käyttäjämäärät kasvavat. Tässä osuudessa sitä tutkitaan. Ensin kuvataan, miten tutkimus on tehty ja miten ratkaisuihin on päädytty. Lopuksi kerrotaan tutkimusten tulokset

4.1 Tutkimusmenetelmät

Tietokannan suorituskyvyn mittaaminen on vaikeaa, koska se riippuu samanaikaisesti monista eri muuttujista. Palvelun kyky käsitellä dynaamisia sivunluonteja riippuu tietokannasta ja tietokantaa käyttävän järjestelmän toteutuksesta. Tästä syystä virtuaalitietokantaa testataan todellisessa työajanseurantajärjestelmässä. Suorituskyky mitataan lisäämällä virtuaalitietokantaan rivejä ja suorittamalla ennalta suunniteltuja hakuja. Rivien lukumäärää lisätään ja samat haut suoritetaan uudelleen. Käytettävien hakujen suunnitellussa tulee ottaa huomioon, että haut olisivat mahdollisimman lähellä todellisia käyttötarpeita. Tässä tutkimuksessa käytettävä satunnaisdata mukailee muodoltaan työajanseurantajärjestelmän käyttämiä työaikoja ja SQL-kyselyt työaikaseurannassa tarvittavia hakuja.

Satunnaisdataa virtuaalitietokantoihin generoidessa mitataan myös tiedon kirjoittamiseen kulunutta aikaa, jotta saadaan selville kirjoitusnopeus. Virtuaalitietokanta on suunniteltu toimimaan single-master-multi-slave -ympäristössä, jolloin kirjoittamisen hitaus voi muodostua pullonkaulaksi, vaikka slave-palvelinten määrää kasvatettaisiinkin kuorman kasvaessa.

Suurin kyselyiden nopeuteen vaikuttava tekijä on tietokannan indeksointi. Indeksointi rakennetaan vastaamaan nimenomaan työajanseurantapalvelun tarpeita.

4.1.1 Tutkimuksessa käytetyt laitteet ja ohjelmistot

Palvelimena käytetään Amazonin Elastic Cloud 2 palvelun large-instanssia[37]. Palvelimessa on 64-bittinen kaksiytiminen Intel Xeon -prosessori. Keskusmuistia on 7,5 Gt ja kiintolevyn koko on 200 Gt. Palvelimen spesifikaatiot ovat samat kuin Movenium Oy:n tuotantopalveluissa käytetyllä tietokantapalvelimella.

Tutkimuksen tekemiseen käytetyt ohjelmistot on lueteltu Kuva 7 taulukossa.

Tehtävä	Ohjelma	Versio
Käyttöjärjestelmä	Ubuntu	10.04.1 LTS
Tietokanta	Percona (MySql)	5.5.30-30.2
HTTP-palvelin	Apache	2.2.14 (Ubuntu)
Ohjelmointikieli	PHP	5.3.2-1ubuntu4.14

Kuva 7. Tutkimuksen tekemiseen käytetyt ohjelmistot.

4.1.2 Tietokannan muoto

Virtuaalitietokantaan luodaan kolme taulua, jotka simuloivat työajanseurantapalvelussa olevan datan muotoa. Ensimmäinen taulu sisältää työajat, toinen projektit ja kolmas käyttäjät. Työajoissa on viittaus asiakkuuteen (partnerid), työn suorittaneeseen käyttäjään sekä projektiin, jota työ koskee. Työaikataulusta indeksoidaan partnerid sekä päivämäärä, koska näitä käytetään hakuehtoina aina työaikoja hakiessa. Lisäkentät 1-8 kuvaavat kenttiä, joiden sisältö on asiakkaan itsensä päätettävissä.

4.1.3 Tiedon kirjoittaminen tietokantaan

Collectorin virtuaalitietokantaan rivejä kirjoitetaan normaalisti Collectorin oman tiedonlisäysfunktion avulla. Funktiolla yhden rivin lisääminen tietokantaan kestää kuitenkin noin 20–50 ms riviä kohden. Funktio on siis liian hidas tutkimustarpeeseen, koska esimerkiksi sadan miljoonan rivin lisääminen saattaisi kestää jopa 2 kuukautta. Tästä syystä rivejä kirjoitetaan tietokantaan tehokkaammalla menetelmällä kuitenkin niin, että tietokannan sisältö simuloi todellista dataa mahdollisimman tarkasti.

[37] Amazonin palvelintyypit. http://aws.amazon.com/ec2/instance-types/.

Virtuaalisessa tietokannassa jokaisen solun sisältö on yksittäinen tietokantarivi, jolloin esimerkiksi yhden työajan lisääminen kirjoittaa tietokantaan useita rivejä eri tauluihin. Suuren data määrän kirjoittaminen tietokantaan toteutetaan lisäämällä rivejä arvioidussa suhteessa erikseen jokaiseen tauluun suurina yksittäisinä komentoina. Esimerkiksi kun lisätään 100 000 työaikaa, todellisuudessa tietokannalle lähetetään komennot lisätä 300 000 päivämäärä- tai kellonaikariviä, 300 000 numerotiedon sisältävää riviä ja niin edelleen.

Tietokantaan kirjoitetaan ensin Collectorin omalla tiedonlisäysfunktiolla 30 työaikaa palveluun, jossa tietokantahaut suoritetaan. Tämän jälkeen tehokkaammalla menetelmällä kirjoitetaan tietokantaan dataa, joka vastaa 100 000 työaikaa, 1000 projektia sekä 100 työntekijää (määrien suhteet ovat arvioita). Tämän jälkeen tietokannan nopeus mitataan ja aloitetaan kierros alusta. Tätä toistetaan, kunnes tietokannassa on noin 150 miljoonaa riviä työaikoja. Tietokannan koko on lopulta noin 100 gigatavua.

4.1.4 SQL-kyselyt

Tutkimukseen käytettävät kyselyt valitaan niin, että ne simuloivat mahdollisimman tarkasti työajanseurantajärjestelmän todellisia hakutarpeita. Erilaisia hakuja suoritetaan kaikkiaan kolme: ensimmäinen hakee yhden kuukauden 100 ensimmäistä työaikaa, toinen hakee myös yhden kuukauden 100 ensimmäistä työaikaa mutta käyttäen hakemiseen vanhempaa, tuotantokäytössä edelleen olevaa hakufunktiota ja kolmas hakee yhden viikon työaikojen summauksen. Kun tietokannan nopeus mitataan, jokainen haku suoritetaan yhteensä 5 kertaa. Hakuaikatuloksista pudotetaan pois lyhyin ja pisin aika, jotta tietokantapalvelimen muiden samanaikaisten prosessien vaikutusta saataisiin minimoitua. Lopuista kolmesta ajasta lasketaan keskiarvo ja tulos tallennetaan tiedostoon. Tuloksista tehdään kuvaajat R-ohjelmointikielen[38] avulla [51].

[38] R on ohjelma statistiikan ja kuvaajien tekemiseen. http://www.r-project.org/.

4.1.4.1 Yhden kuukauden työaikaraportti uutta hakufunktiota käyttäen

Yhden kuukauden työaikaraportin hakeminen tapahtuu Collectorin sisäisellä funktiolla:

```
$data2->cget("worktime",array("date" => "{col} BETWEEN '2013-01-01' AND
'2013-02-01'"), array("limit" => "0,100"));
```

Funktio hakee tietokannasta asiakkaan 100 ensimmäistä työaikaa väliltä 1.1.2013–1.2.2013. Työaikoja haetaan vain 100 ensimmäistä, koska se on suurin rivimäärä, mitä raporteilla näytetään palvelussa kerralla. Haku on raskas, koska se palauttaa työaikojen kaikki sarakkeet.

Uuden ja vanhan hakufunktion välillä tärkein nopeuteen vaikuttavaa tekijä on uuden hakufunktion käyttämä, niin sanottu viivästetyn haun tekniikka [52]. Kun tietokantahaussa käytetään runsaasti JOIN-tekniikka (kuten Collector tekee), suorittaa MySql haun tehottomasti. MySql valitsee ensin tietokannasta rivit eniten rajoittavalla ehdolla ja tämän jälkeen yhdistää kaikki JOIN-liitetyt taulut haettuihin riveihin. Kaikki liitetyt taulut yhdistetään siis ensin rajattuihin riveihin (joita voi olla tuhansia) vaikka haussa lopulta rajoitetaankin tulos sataan riviin. Tehokas tapa suorittaa haku olisi ensin rajata haettavat rivit eniten rajoittavan ehdon mukaan. Tämän jälkeen riveihin liitetään vain taulut, joiden sisältöä käytetään datan järjestämiseen ja rajoittamiseen. Vasta kun 100 riviä on valittu, liitetään kaikki muut taulut. Tätä MySql ei kuitenkaan itse osaa tehdä. Tämän takia uuden funktion haku perustuu tekniikkaan, jossa yksittäinen haku tehdään suorittamalla kaksi peräkkäistä hakua. Ensin haetaan taulut, joita tarvitaan rivien rajoittamiseen. Tulos tallennetaan väliaikaistauluun. Väliaikaistauluihin tulee siis aina maksimissaan 100 riviä. Tämän jälkeen tehdään toinen haku, jolla kaikki loput taulut liitetään väliaikaistaulun riveihin.

4.1.4.2 Yhden kuukauden työaikaraportti vanhaa hakufunktiota käyttäen

Yhden kuukauden työaikaraportin hakeminen vanhalla funktiolla suoritetaan komennolla:

```
$data->cget("worktime",array("{date} BETWEEN '2013-01-01' AND '2013-02-
01'"), array("limit" => "0,100"));
```

Haku on muuten identtinen edellisen kanssa, mutta tässä käytetään vanhempaa funktiota. Vanhemman funktion tutkiminen on tärkeää, koska sitä käytetään edelleen pääasiassa kaikissa tuotantopalveluissa. Vanhan funktion tehokkuus tiedetään huonommaksi, mutta tutkimuksella on tarkoitus selvittää funktion toimivuus tulevien vuosien aikana. Koska funktiota käytetään tällä hetkellä tuotannossa olevissa palveluissa, tiedetään että funktion tehokkuus riittää vielä sillä data määrällä, joka tuotanto palveluiden tietokanta tällä hetkellä sisältää. Funktion toiminnasta tietomäärän kasvaessa ei kuitenkaan ole varmuutta ja tämän vuoksi asiaa tutkitaan. Funktion korvaaminen kokonaan uudemmalla versiolla on haastavaa, koska funktiot eivät ole täysin yhteensopivia. Tästä syystä funktion vaihtamiseen ei todennäköisesti aleta, jollei siihen ole pakottavaa tarvetta.

4.1.4.3 Yhden työviikon summaushaku

Yhden työviikon summat haetaan Collectorin sisäisellä funktiolla seuraavasti:

```
$data2->cget("worktime",array("date" => "{col} BETWEEN '2013-01-01' AND
'2013-01-07'"),array("group" => array("user")));
```

Funktio hakee tietokannasta asiakkaan kaikki työajat väliltä 1.1.2013 – 1.7.2013 työntekijöittäin summattuna.

4.1.5 Palvelun testaaminen suuressa tietokannassa

Kun testitietokantaan on kirjoitettu 150 miljoonaa työaikaa, testataan tietokannassa olevaa palvelua normaalissa käyttötilanteessa. Suuren tietokannan käyttäminen voi tuoda esille niin sanottuja pullonkauloja, jos järjestelmässä sellaisia on. Pullonkaulat aiheutuvat huonosti suunnitelluista tietokantahauista. Jos lähdekoodissa on yksikin tietokantahaku, jonka suorittaminen hidastuu nopeasti tietomäärän kasvaessa, voi se estää koko palvelun toiminnan. Pullonkaulat eivät kuitenkaan tule välttämättä tuotantokäytössä itsestään esille, koska huonostikin toteutettu tietokantahaku voi toimia tarpeeksi nopeasti niin kauan, kun tietokanta pysyy kohtuullisen pienenä.

Testaaminen toteutetaan käyttämällä MySql:n log-slow-queries toimintoa ja asettamalla sen aikarajaksi 1 sekunti, jolloin MySql tallentaa lokitiedostoon kaikki SQL-komennot, joiden suorittaminen kestää yli sekunnin. Pienemmänkin aikarajan käyttäminen olisi mahdollista, mutta yksi sekunti on hyvä kompromissi pullonkaulojen paikallistamiseen. Palvelua käytetään selaimella mahdollisimman monipuolisesti eli katsellaan erilaisia raportteja ja lisätään työaikoja ja käyttäjiä.

4.1.6 Testien suorittaminen

Kaikki testit tehdään suorittamalla testiohjelma palvelimella. Ohjelman lähdekoodi on annettu liitteenä. Ohjelman alussa Collector alustetaan liittämällä tiedosto include.php (tämän tiedoston sisältö on Movenium Oy:n omaisuutta). Tiedosto alustaa käytettäväksi Collectorin sisältämät funktiot ja avaa tietokantayhteyden. Seuraavana ohjelmassa määritellään vakiota, kuten kuinka monta työaikamerkintää tietokantaan kirjoitetaan päivää kohden sekä kuinka monta kierrosta ohjelmaa ajetaan.

Tämän jälkeen ohjelma suorittaa silmukkaa, jonka aikana lisätään yksi työaika Collectorin omalla lisäysfunktiolla sekä täytedataa add_bulk_data metodin avulla. Täytedata on muodoltaan mahdollisimman samanlaista kuin tietokannassa olevat työajat, mutta niiden kirjoittaminen tietokantaan on tehokkaampaa. Täytedataa kirjoitetaan joka kierroksella yhteensä noin 100 000 riviä. Viimeisenä silmukan lopussa suoritetaan funktiot measure_select_speed, measure_select_speed_week_sum ja measure_select_speed_old_get_function. Ne suorittavat aiemmin kuvattuja hakukyselyitä ja tallentavat hakuajat tiedostoon.

4.1.7 Tulosten vertaaminen tuotantotietokantaan

Jotta tulosten oikeellisuutta voitaisiin tutkia, verrataan mittaustuloksia todellista dataa vastaan. Samoja hakuja, joita suoritettiin virtuaalitietokannan tutkimisessa, suoritetaan tietokannassa, joka on kopio tuotantopalvelussa olevasta tietokannasta. Näin voidaan varmistua siitä, että testitietokanta, johon data on kirjoitettu apufunktioilla, käyttäytyy samalla tavalla kuin todellista dataa sisältävä tietokanta.

Tietokantana käytetään tuotantokopiota päivältä 26.6.2013. Tuotantotietokannan kopiossa suoritetaan täsmälleen samat haut kuin aiemmissa tietokantatesteissä.

4.1.8 Tuotantotietokannan kasvunopeuden arviointi

Tutkimuksen testituloksilla ei ole kovinkaan suurta merkitystä, jollei pyritä myös arvioimaan, kuinka nopeasti tuotannon tietokanta tulee kasvamaan yhtä suureksi kuin tutkimuksessa käytetty testitietokanta. Tutkimukseen tarvittavat kertoimet lasketaan samasta tuotantokopiosta, jota käytettiin tulosten vertailuhakujen suorittamiseen.

Kuukauden sisällä tietokantaan kirjoitettavan datan määrää voidaan arvioida kaavalla (1).

$$Tilan\ tarve\ kuukaudessa\ [tavua] = L_{palvelut} \times K \times T_{kk} \times \alpha \qquad (1)$$

$L_{palvelut}$ = Palveluiden lukumäärä
K = Käyttäjien keskimääräinen lukumäärä per palvelu
T_{kk} = Yhden käyttäjän kuukaudessa syöttämien rivien keskimäärä
α = Yhden tietokantarivin keskimääräinen tilantarve tavuina

Arvojen K, T_{kk} ja α oletetaan pysyvän vakioina ja niiden suuruudet lasketaan tuotantodatan perusteella. K lasketaan jakamalla tuotantokopiosta löytyvien käyttäjien lukumäärä palveluiden lukumäärällä. Arvio ei ole täysin tarkka, koska palveluiden lukumäärä on muuttunut tarkasteluvälin aikana, mutta tästä aiheutuva virhe oletetaan pieneksi. T_{kk} lasketaan jakamalla kuukauden sisällä lisättyjen rivien lukumäärä kokonaiskäyttäjämäärällä. Vakio α lasketaan jakamalla koko tietokannan kiintolevyllä käyttämä tila tavuissa tietokannan sisältämällä rivimäärällä. Näin laskettu tilantarve sisältää siis sekä itse datan käyttämän tilan että indekseihin tarvittavan tilan. Nämä kolme vakiota voidaan yhdistää kaavan (2) mukaan yhdeksi vakioksi. Kaavan avulla lasketaan vakio γ, joka kuvaa yhden palvelun keskimääräistä tilantarvetta kuukaudessa.

$$\gamma = K \times T_{kk} \times \alpha \quad (2)$$

K= Käyttäjien keskimääräinen lukumäärä per palvelu
T_{kk} = Yhden käyttäjän kuukaudessa syöttämien rivien keskimäärä
α = Yhden tietokantarivin keskimääräinen tilantarve tavuina

Muuttuja $L_{palvelut}$ ei ole vakio vaan sen oletetaan kasvavan joka kuukausi. Kasvunopeus riippuu yrityksen kasvunopeudesta. Jos yrityksen koko pysyisi vakiona, muuttujan $L_{palvelut}$ oletetaan kasvavan lineaarisesti. Jos yritys kasvaa, oletetaan muuttujan $L_{palvelut}$ kasvavan yrityksen kasvunopeuteen nähden eksponentiaalisesti.

$$L^k_{palvelut} = a\,(1 + \frac{k}{12})^n \quad (3)$$

$L^k_{palvelut}$ = Palveluiden lukumäärä k-kuukauden kuluttua
a = Palveluiden lukumäärän alkuarvo
k = Aika alkuhetkestä kuukausina
n = Yrityksen kasvunopeuskerroin vuodessa

Kaavassa (3) on kuvattu, miten muuttuja $L^k_{palvelut}$ lasketaan. Yrityksen kasvunopeuskertoimen vuodessa (n) oletetaan olevan välillä [1,2], jossa 1 = ”yritys ei kasva lainkaan” ja 2 = ”Yrityksen koko kaksinkertaistuu vuosittain”. Näistä voidaan johtaa kaava (4), jolla saadaan laskettua tietokannan kokonaistilantarve tietyn aikavälin kuluttua.

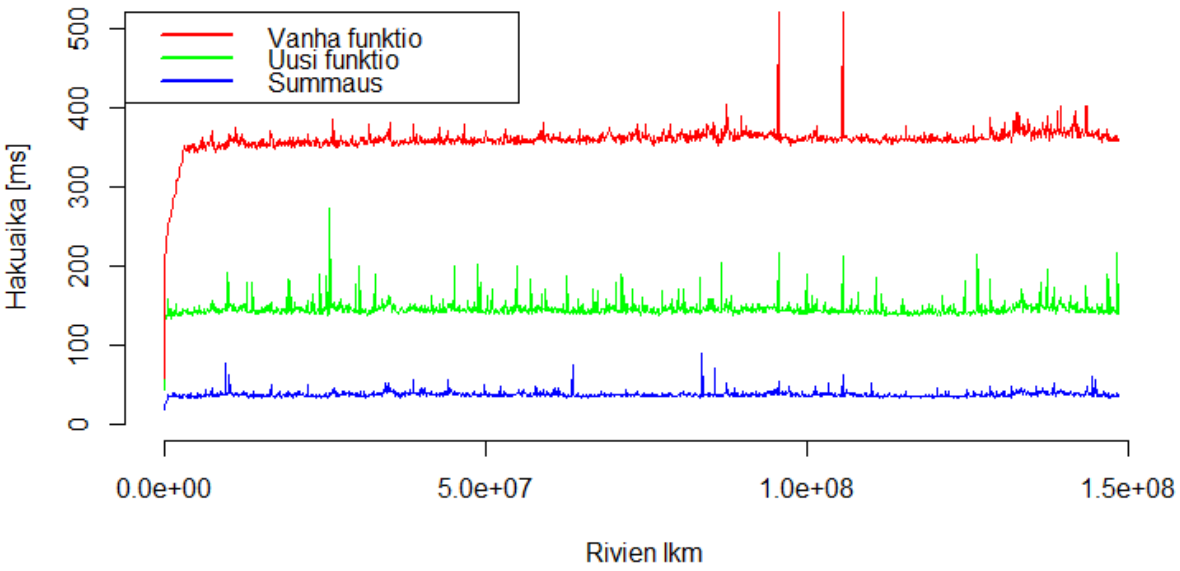
$$Tietokannan\ vaatima\ tila = S_{db} = a \times \gamma \times \sum_{k=1}^{t} (1 + \frac{k}{12})^n \quad (4)$$

a = Palveluiden lukumäärän alkuarvo
γ = Palvelun vaatima keskimääräinen tilantarve kuukaudessa
$t = Tarkasteluväli\ kuukausina$
k = Aika alkuhetkestä kuukausina
n = Yrityksen kasvunopeuskerroin vuodessa

Kaavan 4 avulla voidaan siis laskea tietokannan vaatima tilantarve. Kaavan vakiot lasketaan tutkimalla tuotantotietokantaa. Tuotantotietokannasta tutkitaan kuinka monta palvelua ja käyttäjää se sisältää, sekä tietokannan kokonaistilantarve kiintolevyllä. Näiden tietojen ja yllä esitettyjen kaavojen 1-4 avulla voidaan arvioida tietokannan tilantarvetta tulevaisuudessa.

4.2 Tulokset ja niiden arviointi

Kuva 8 on esitetty, miten tietokannan kokonaisrivimäärä vaikuttaa tietokannassa suoritettavien hakujen suoritusaikaan. Kuvassa on kolme kuvaajaa, joista jokainen edustaa eri hakufunktion suoritusaikaa. X-akseli esittää tietokannan sisältämää rivimäärää. Kuva 9 oleva kuvaaja esittää, miten tietokannan koko vaikuttaa tietokantaan kirjoittamisen nopeuteen.

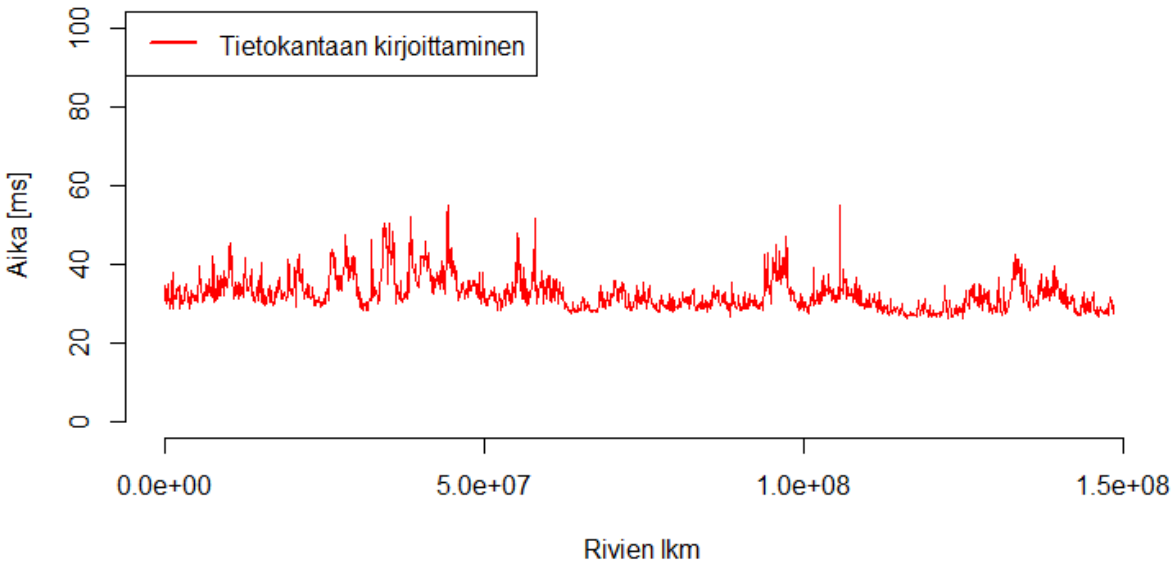


Kuva 8. Hakuajat rivimäärän kasvaessa

Kuvista 8 ja 9 voidaan nähdä, että tietokannan sisältämän datan määrällä ei ollut merkittävää vaikutusta tietokantahakujen tai tietokantaan kirjoittamiseen kuluneen ajan kestoon. Tulos on odotusten mukainen. Kun tietokantahaut tehdään oikein eli indeksien avulla, tietokannan datamäärällä ei ole juurikaan vaikutusta hakuaikoihin. Ei varsinkaan silloin, kun dataa haetaan melko vähän (vain 100 riviä) tai kun summauksessa ei summata suurta määrää rivejä. Todellisissa työajanseurannan hakutarpeissa tämä

pitää lähes aina paikkaansa. Poikkeuksena ovat suurien asiakkaiden pitkän aikavälin summat, joissa saatetaan tarkoituksella summata tuhansia tai kymmeniä tuhansia työaikoja yhteen.

Yhden kuukauden työaikojen (100 ensimmäistä) hakuaika oli uutta hakufunktiota käyttäen keskimäärin noin 146 ms. Tästä voidaan laskea, että keskimäärin yksittäinen tietokantapalvelin pystyy tuottamaan karkeasti noin 400 työaikaraporttia minuutissa. Sama haku suoritettuna vanhemmalla funktiolla kesti keskimäärin noin 359 ms. Vanhan funktion suoritusaika on siis noin 2,5 -kertainen uuden funktion hakuaikaan verrattuna.



Kuva 9. Kirjoitusnopeus rivimäärän kasvaessa

Viikon työaikojen summaaminen vei aikaa keskimäärin noin 38 ms. Summaamisen nopeutta selittää kuukauden työaikojen hakemista pienempi rivimäärä (noin 200). Myös siirrettävä tietomäärä palvelimien välillä on pienempi, kun työaika rivit on summattuna.

Tutkimukseen käytetty palvelin on identtinen Movenium Oy:n tuotantopalvelimien kanssa. Näin ollen tutkimustuloksista voidaan päätellä että tietokantaratkaisu säilyy riittävän tehokkaana ainakin tietokannan kasvaessa noin 100 Gt suureksi. Palvelimen

keskusmuistin määrä vaikuttaa paljon tietokannan tehokkuuteen. Kun haun tarvitsemat väliaikaistaulut eivät enää mahdu keskusmuistiin, hakujen tehokkuus laskee tuntuvasti (useita kertaluokkia). Tuon kriittisen rajan arvioiminen voi olla vaikeaa. Tästä syystä tutkimuksen antamat tulokset ovat ainoa toimiva tapa päätellä, että myös tuotantopalvelimien keskusmuisti riittää ainakin niin kauan, että ylitetään tuo 100 gigatavun tietokantakoko.

Tutkimuksessa ei otettu kantaa siihen, miten paljon käyttäjäkuormaa tietokantaratkaisu kestää. Rajaksi voi tulla esimerkiksi samanaikaisesti auki olevien tietokantayhteyksien lukumäärä. Käytössä oleva master-slave -replikointi kuitenkin takaa sen, että jos palvelimien kapasiteetti ei riitä tuleviin tietokantapyyntöihin, voidaan slave-palvelimien lukumäärää kasvattaa teoreettisesti rajatta. Työajan kirjoittaminen tietokantaan vei aikaa keskimäärin noin 32 ms. Tämä on ainoa toiminne, joka käyttää master-palvelinta. Yksinkertaistetusti laskien voidaan siis arvioida, että tietokantaratkaisuun voidaan kirjoittaa vuorokaudessa noin 3 miljoonaa riviä uutta tietoa.

4.2.1 Hitaiden tietokantahakujen paikallistaminen

Kun käytettiin palvelua suurella testitietokannalla, löytyi useampi yli sekunnin kestävä haku. Hitain löydetty haku kesti tietokannassa yli 300 sekuntia. Kun ei oteta huomioon näitä hitaita hakuja, oli yleistuntuma palvelua käyttäessä, että tietokannan suuri koko ei hidastanut järjestelmän toimintaa.

4.2.1.1 Lista löydetyistä hitaista hauista

Asiakkaiden haku: Haku, jonka avulla tietokannasta palautetaan kaikki tietokannassa olevat palvelut. Haku kesti yli 300 sekuntia. Tämä johtuu siitä, että palveluita haetaan tietokannasta ilman asiakastietoa (partnerid) eikä indeksointia voida tällöin käyttää. Ongelma voidaan korjata esimerkiksi lisäämällä indeksi, joka ei sisällä asiakastietoa.

Työviikon haku: Palvelun etusivulla on näkymä, jossa näkyy kirjautuneen käyttäjän kuluvan viikon työajat. Työajat haetaan jostain tuntemattomasta syystä (todennäköisesti epähuomiossa) ilman asiakastietoa, jolloin indeksin käyttö ei ole

mahdollista ja hakuaika on yli 26 sekuntia. Ongelma voidaan korjata muuttamalla haku järkevämmäksi.

Viimeisimmät lisätyt rivit: Palvelun etusivulla on listattuna palveluun viimeisimpänä lisätyt uudet rivit. Haku ei pysty käyttämään taululla olevia indeksejä. Tästä syystä haun suoritusaika on yli 3 sekuntia. Vika voidaan korjata muotoilemalla haku uudelleen ja mahdollisesti lisäämällä sopiva indeksi tietokantaan.

Työaikaraportin oletusnäkymä: Kun käyttäjä menee ensimmäistä kertaa työaikaraportille, haussa on oletuksena päivämääräväli ilman lopetuspäivämäärää. Tästä syystä oletusraportin haku kesti isossa tietokannassa yli 300 sekuntia vaikka tietokannasta palautui vain 100 riviä. Tämä ongelma on korjattavissa estämällä käyttäjää hakemasta työaikoja rajoittamattoman suurilla päivämääräväleillä.

Raportin vienti tiedostoon/lähettäminen sähköpostiin: Kun esimerkiksi työaikaraportti on erittäin pitkä (yli 1000 riviä), sen hakeminen ei kestä selaimeen liian kauan sivutuksen ansiosta, koska silloin kerralla tietokannasta haetaan maksimissaan 100 riviä (yksi sivullinen). Kun raportti sitten viedään esimerkiksi PDF-tiedostoon, joudutaan kaikki rivit hakemaan tietokannasta kerrallaan. Koska rivimäärää ei ole mitenkään rajattu, voi kyseinen haku kestää todella pitkään (testissä noin 150 sekuntia).

Tämä voidaan korjata asettamalla joku sopiva raja sille, miten paljon tietoa voidaan viedä esimerkiksi tiedostoon. Jos loppukäyttäjät tarvitsevat aidosti pidempiä raportteja, ja siksi rajaaminen ei ole mahdollista, täytyy suuria hakuja varten käynnistää oma replikoitu palvelimensa.

Sekalaisia hitaita hakuja: Muutamat eri haut kestivät isoa tietokantaa käyttäessä pitkään, koska tietokanta aloitti haun suorittamisen eri järjestyksessä kuin on suunniteltu. Tästä syystä muutamat haut kestivät noin 3 sekuntia vaikka niiden tulisikin päättyä vähintäänkin alle puolessa sekunnissa. Ongelman ratkaisu on indeksin lisääminen tai olemassa olevan indeksin muuttaminen siten, että indeksoituna on sekä

rivin päivämäärä, että lomakkeen tunniste. Tällä hetkellä indeksoituna on ainoastaan rivin päivämäärä ja tästä syystä indeksi ei ole tarpeeksi kattava.

4.2.2 Tuotantotietokannan kopiolla suoritetut kontrollihaut

Palvelun koko (käyttäjämäärä)	**11**	**18**	**32**	**38**	**46**	**108**
Palvelussa työaikoja [kpl]	3378	6370	3742	1587	6675	39470
Keskimääräinen työajat/kuukausi [kpl]	119	155	269	112	476	1400
Hakuaika: kuukauden työajat [ms]	183	228	218	144	181	385
Hakuaika: kuukauden työajat (vanha funktio) [ms]	379	513	358	318	449	1566
Hakuaika: viikon työaikojen summaus [ms]	88	69	63	37	213	720
Työaikatiedon sarakemäärä [kpl]	12	21	15	11	11	24

Kuva 10. Taulukossa on esitetty tuotantotietokannan kopiolla tehtyjen hakujen tulokset. Käytetyt palvelut on valittu satunnaisesti.

Kuva 10 oleva taulukko sisältää testeissä käytettyjen hakujen suoritusajat tuotantotietokannassa. Tuotannossa olevasta tietokannasta on otettu kopio, jossa haut on sitten suoritettu. Jokainen haku on suoritettu 10 kertaa ja suoritusajoista on laskettu keskiarvot. Kaikki haut on suoritettu kuudessa eri palvelussa. Palvelut on valittu satunnaisesti, mutta kuitenkin siten, että ne edustaisivat erikokoisia palveluita. Palvelun koon mittana on käytetty palveluun lisättyjen käyttäjien lukumäärää. Taulukon alin rivi, Työaikatiedon sarakemäärä, kertoo kuinka monta saraketta kyseisessä palvelussa on liitetty työaikatietoihin. Sarakkeiden lukumäärä on vaikuttaa haun kestoon, koska

Collector liittää jokaisen sarakkeen JOIN-komennolla varsinaiseen hakuun. Näin ollen hakuaika kasvaa sarakemäärän kasvaessa.

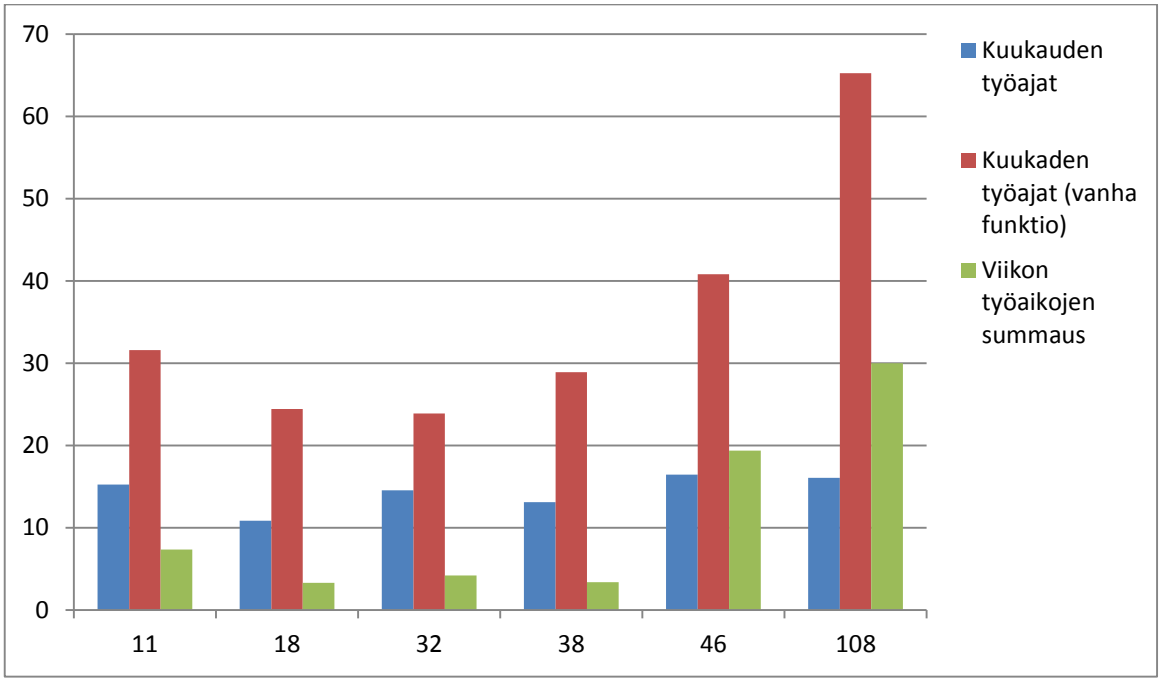
Jos suurinta palvelua (108 käyttäjää) ei oteta huomioon (syy tähän on selitetty myöhemmin), Kuva 10 olevasta taulukosta laskemalla saadaan keskimääräisiksi hakuajoiksi: kuukauden työajat 191 ms, kuukauden työajat vanhaa funktiota käyttäen 403 ms ja viikon työaikojen summaus 94 ms. Hakuajat olivat noin 40–60 ms pidempiä testitietokantaan verrattuna. Pidemmät hakuajat selittyvät suurimmaksi osaksi sillä, että todellinen data on kiintolevyllä hajautuneempaa. Testitietokannassa työajat kirjoitettiin tietokantaan fyysisesti peräkkäin, jolloin hakua suorittaessa kiintolevyn I/O-nopeus ei vaikuta hakuun yhtä paljon kuin tuotantodatan kanssa. Tuotantodatassa työajat ovat fyysisesti hajaantuneet enemmän kiintolevylle, jonka takia kiintolevyn lukupää joutuu vaihtamaan paikkaa enemmän. Tämän perusteella voidaan sanoa, että testitietokannan kanssa saadut tulokset ovat vertailukelpoisia todellisen datan kanssa, koska hakuaikojen absoluuttiset arvot ovat samaa suuruusluokkaa. Tästä voidaan päätellä, että tietokannan hakuajat pysyvät siedettävinä myös tuotantodataa sisältävän tietokannan kasvettua noin 100 gigatavuiseksi.

Suurimman palvelun (108 käyttäjää) hakuajat ovat huomattavasti suuremmat kuin muissa palveluissa. Tämä johtuu suuremman tietomäärän lisäksi myös työaikatiedon suuresta sarakemäärästä. Palvelussa on työajoilla yhteensä 21 saraketta. Tietokannan rakenteen takia jokainen sarake liitetään erikseen lopulliseen hakutulokseen. Tästä syystä hakuaika kasvaa lähes lineaarisesti suhteessa sarakkeiden lukumäärään.

Kuva 11 on esitetty samat tulokset kuin Kuva 10 taulukossa sillä erolla, että hakuajat on jaettu palvelun työaikatiedon sarakemäärällä. Kuvassa X-akselilla on kaikki kuusi eri palvelua, joissa haut on suoritettu. Palveluihin viitataan palvelussa olevien käyttäjien lukumäärän mukaan, kuten Kuva 10. Kuvassa Y-akselin arvot ovat hakujen suoritusaikoja normalisoituna palvelussa työaikatietoon liittyvien sarakkeiden lukumäärän suhteen.

Kuva 11 histogrammista nähdään, että normalisoituna kuukauden työaikojen hakemiseen kuluva aika uudella funktiolla on kaikissa palveluissa lähes sama. Vanhalla

funktiolla suoritettu kuukauden työaikojen hakemiseen kulunut aika kasvaa tuntuvasti, mitä isommasta palvelusta on kyse. Tämä johtuu siitä, että vanhassa funktiossa tietokanta joutuu ensin hakemaan kaikki hakuvälin rivit ennen kuin se voi rajata hakutuloksen. Tällöin hakuaika kasvaa lineaarisesti suhteessa palvelussa olevien työaikojen lukumäärään.



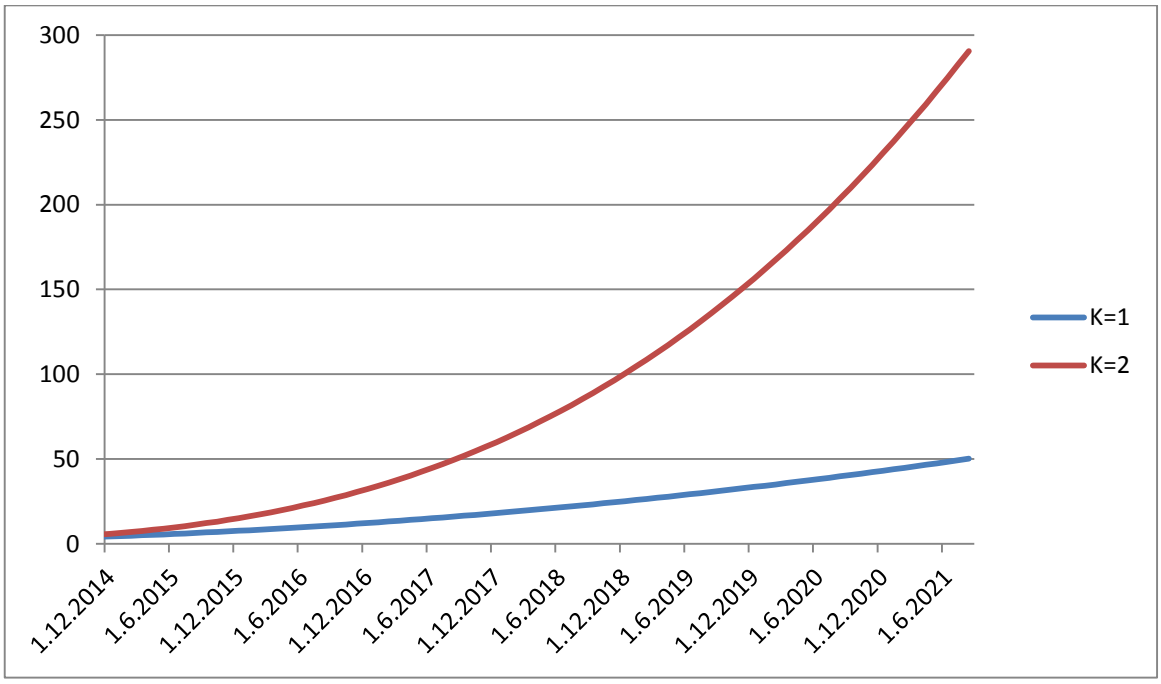
Kuva 11. Tuotantotietokannan kopiolla suoritettujen hakujen normalisoidut tulokset. Hakuaika [ms] on jaettu työaikatiedon sarakkeiden lukumäärällä.

Tuloksista havaitaan, että haku ”Kuukauden työajat (vanha funktio)” suoritusaika on suurempi palveluissa, joiden käyttäjämäärät ovat 11 ja 18 kuin palvelussa, jonka käyttäjämäärä on 32. Tällä ei kuitenkaan ole havaittavissa suoraa selitystä. Ainoa perustelu hakuajoille on, että tuotanto dataa käytettäessä hakuaikoihin vaikuttaa paljon satunnaisia tekijöitä. Hakuaikaan voi vaikuttaa esimerkiksi se, miten fragmentoituneesti [53] hakuun sisältyvät tietokantarivit ovat kiintolevyllä. Näin ollen esimerkiksi palvelun ikä voi vaikuttaa hakuaikoihin, koska vanhemman palvelun tietokantarivit sijaitsevat kiitolevyllä suuremmalla alueella. Tämä ei yksin riitä selittämään hakuaikoja, mutta on esimerkki siitä, että eri palveluissa hakuaikojen suoritusajoissa on myös satunnaisuutta.

Hakuajoissa on eroavaisuuksia etenkin kun käytettävä data on peräisin tuotannossa olevasta palvelusta.

Kuva 11 nähdään, että viikon työaikojen summaus kasvaa selvästi sen mukaan, mitä suuremmasta palvelusta on kyse. Tämä selittyy yksinkertaisesti sillä, että työaikojen lukumäärän kasvaessa on enemmän summattavaa. Ensin kaikki hakuaikavälin työajat täytyy hakea tietokannasta ja tämän jälkeen laskea yhteen.

4.2.3 Tuotantotietokannan kasvunopeuden arvioinnin tulokset



Kuva 12. Kuvaajassa on esitetty tuotantotietokannan koko gigatavuina kahdella yrityksen kasvunopeudella laskettuna.

Tuotantotietokannan kopiosta tutkimalla saadaan kappaleessa 4.1.8 esitetyn kaavan (2) perusteella laskettua, että $\gamma \approx 110600$ tavua/kk/palvelu.

Vakion laskentaan käytettyjä numeroarvoja ei esitetä, koska ne sisältävät Movenium Oy:n kannalta luottamuksellista tietoa.

Kuva 12 esitetty tietokannan tilantarve on laskettu kaavan 4 mukaisesti. Kuvasta nähdään, että jos yritys kasvaa nopeudella K=2, saavutetaan tutkimuksessa käytetty tietokannan koko (noin 100 Gt) vuoden 2018 lopulla. Kasvunopeudella K=1, sadan gigatavun kokoa ei saavuteta tarkasteluaikavälillä lainkaan. Koska yrityksen kasvunopeus oletettiin olevan näiden arvojen välissä, todellinen tietokannan tilantarve on jotain näiden kahden kuvaajan väliltä. Tämän tarkempaa arviota tilan tarpeesta ei pystytä tekemään, koska yrityksen todellista kasvua ei voida ennustaa.

5 Yhteenveto

Tässä työssä tutkittiin Movenium Oy:n kehittämää Collector-nimistä virtuaalitietokantaratkaisua työajanseurantapalvelun alustana. Collector on MySql:n päälle rakennettu virtualisointi, jonka tarkoituksena on mahdollistaa palveluiden helppo aseteltavuus ja räätälöitävyys. Sitä verrattiin muihin olemassa oleviin ratkaisuihin ja tutkittiin sen tehokkuutta palvelun alustana tietomäärän kasvaessa.

Collectoria verrattiin useisiin eri ratkaisuihin. Perinteisen relaatiotietokannan ongelma työajanseurannan tarpeisiin on sen jäykkyys. Relaatiotietokannan taulujen rakenne ei ole joustava, vaan taulujen sisältö on aina siinä muodossa miten tietokannan taulut on määritelty. Asiakkaiden palveluiden eroavaisuudet on vaikea toteuttaa, jos datamalli on kaikilla asiakkailla oltava identtinen.

NoSql-tietokannat, kuten avoimeen lähdekoodiin perustuvat MongoDB ja Facebookin alun perin kehittämä Cassandra, mahdollistavat paremmin erilaisten datamallien käsittelyn yhteisessä tietokannassa. Niitäkään ei ole kuitenkaan varsinaisesti kehitetty siihen tarpeeseen, että palveluiden datamallit vaihtelevat asiakkaittain. NoSql-tietokantojen suurimpana etuna on niiden erittäin tehokas suorituskyky myös todella suurten (satoja gigatavuja isojen tietokantojen) tietomäärien käsittelyssä. Niiden suorituskyky perustuu tiedon tarkkuudessa tehtäviin myönnytyksiin. NoSql-tietokannan sisältö ei ole aina täysin päivittynyt, vaan se päivittyy viivästyneesti vasta jonkin ajan kuluessa.

Esteitä NoSql-tekniikan käyttöönottamiseen on yleensä kaksi. Ensimmäinen niistä on jo olemassa olevan tietokantarakenteen sovittaminen NoSql-tietokannan muotoon. Toinen ongelma on SQL-kielen puuttuminen. Lähes kaikki ohjelmoijat ovat tottuneet SQL-kielen käyttöön ja näin ollen NoSql:n käyttäminen vaatii aluksi paljon oppimista, kun taas relaatiotietokannan yksinkertaiseen käyttöön tarvittava tietotaito on yleensä jo valmiina. SQL-kielen puute tekee myös erilaisten raporttien toteuttamisen usein vaikeammaksi, koska SQL-kieli itsessään sisältää valmiina suuren joukon ominaisuuksia, joilla raportteja voidaan toteuttaa varmalla ja yksinkertaisella tavalla.

Tietokanta palveluna (DBaaS) -mallissa yritys ostaa tietokantaratkaisunsa valmiina palveluna. Näin yritys säästyy tietokannan skaalaamisen ja muun ylläpidon kustannuksilta. Ainakin tällä hetkellä DBaaS-palvelut tarjoavat lähinnä NoSql -tyyppisiä tietokantoja. Relaatiotietokannan tarjoaminen palveluna on hankalaa relaatiotietokannan skaalautuvuusongelmien takia. Jos tarjolla olisi edullisesti skaalautuva ja tarpeeksi suureen tehokkuuteen kykenevä relaatiotietokantapalvelu, voisi myös olla mahdollisesti kustannustehokasta käyttää Collector-järjestelmää sellaisen päällä.

Myös Ruby on Rails -ohjelmointitekniikkaa tarkasteltiin, mutta se ei itsessään tarjoa helpotusta asiakastarpeesta johtuvaan palveluiden erilaisuusongelmaan. Tulevaisuuden kehitystyön suunnittelussa Moveniumin tulisi tarkastella enemmän Ruby on Railsin käyttämää MVC-mallia (model-view-control), jossa järjestelmän logiikka ja käyttöliittymä eriytetään mahdollisimman täydellisesti toisistaan. Näin ohjelman rakenne ja kehitettävyys pysyvät hyvänä myös lisäkehitystä tehdessä.

SaaS-ohjelmiston kuorman siirtämisellä käyttäjien päätelaitteille, voidaan palvelimien kuormaa vähentää. Tähän tarkoitukseen soveltuvia Javascript-ohjelmistoalustoja esiteltiin kaksi. AngularJS sekä Ember.js ovat molemmat tarkoitettu internet-selaimessa toimivien sovellusten kehittämiseen. Molemmat näistä alustoista hyödyntävät MVC-mallia. Järjestelmä voidaan toteuttaa siten, että päätelaiteella suoritettava ohjelma sisältää järjestelmän käyttöliittymän ja palvelin toimii datavarastona. Näin ohjelman toiminta saadaan jaettua kahteen erilliseen osa-alueeseen, ja ohjelman arkkitehtuuria saadaan selkeytettyä.

Työn tutkimusosuus tarkasteli Collectorin tehokkuutta ja toimintaa tulevaisuudessa, kun asiakasmäärä ja tietokannan koko kasvavat. Tutkimuksessa testitietokannan koko kasvatettiin keinotekoisesti noin 100 gigatavun kokoiseksi, jonka aikana tietokannassa suoritettiin työajanseurantapalvelun tarvitsemia tietokantahakuja ja tutkittiin näiden suoritusaikoja suhteessa tietokannan kokoon.

Odotetusti hakuajat eivät kasvaneet lainkaan tietomäärän kasvaessa, koska oikein indeksoitujen hakujen nopeuteen ei tietokannan koolla tulisikaan olla juuri vaikutusta. Tietokannan kasvaessa tarpeeksi suureksi, keskusmuistin määrä olisi voinut hidastaa hakuja tuntuvasti. Koska näin ei kuitenkaan käynyt voidaan päätellä, että Moveniumin tuotantopalvelimilla oleva keskusmuistin määrä riittää ainakin tutkitun kokoisen tietokannan kanssa.

Suuressa tietokannassa palvelua käytettiin normaalisti internet-selaimen avulla ja paikallistettiin useampi ongelmakohta, jonka aiheuttaja on jossakin tietokantakyselyssä oleva pullonkaula. Pullonkaulojen olemassaolo ei tällä hetkellä vielä juurikaan näy tuotantopalvelimien kuormassa, mutta niiden vaikutus tulee olemaan suuri tietomäärän kasvaessa. Näihin pullonkauloja aiheuttaviin tietokantakyselyihin esitettiin helppoja ratkaisuohjeita.

Palveluun saattaa jäädä pullonkauloja tässä työssä esitettyjen lisäksi. Jos kaikki mahdolliset pullonkaulat haluttaisiin varmuudella löytää, täytyisi tutkimukseen käyttää parempia lähestymistapoja kuin pelkkä palvelun manuaalinen läpikäyminen.

Tutkimuksessa saatuja tuloksia verrattiin tuotantotietokannan kopion tuloksiin ja todettiin, että todellisen datan kanssa hakuajat olivat noin kaksinkertaisia. Tämä ei kuitenkaan tarkoita, että tutkimustulokset olisivat epäonnistuneita, vaan että todellisen datan kanssa kiintolevyn I/O-nopeus hidastaa hakua, koska todellinen data on kiintolevyllä hajautuneemmin kuin testitietokannan järjestyksessä kirjoitettu data. Tulokset osoittavat kuitenkin sen, että hakuajat eivät kasva tietomäärän kasvaessa. Eivät ainakaan alle 100 Gt tietokannassa.

Ongelma tämän tyyppisessä tutkimisessa on, että todellinen tietokantaan kertyvä data voi käyttäytyä odottamattomasti. Koska generoidulla datalla ja tuotantotietokannan kopiolla tehtyjen hakujen suoritusajat olivat toisistaan poikkeavia, ei voida varmasti sanoa pitääkö tämän tutkimuksen testitulokset paikkaansa myös kun tietokannassa on 100 Gt loppukäyttäjien lisäämää dataa. Myös tietokantapalvelimen suurempi kuormittaminen voi vaikuttaa suoritusaikoihin.

Lopuksi arvioitiin vielä sitä, miten nopeasti tietokannan koko tulee kasvamaan, jos Movenium Oy kasvaa ennustetulla tavalla. Oletuksena oli, että Movenium Oy:n kasvunopeus on välillä, yritys ei kasva lainkaan ja yritys kasvaa 100 % vuodessa. Tällä oletuksella saatiin arvioitua, että 100 gigatavun tietokanta koko saavutetaan aikaisintaan vuoden 2018 lopulla.

Tietokannan kasvunopeuden arvioinnissa ei otettu huomioon lainkaan asiakkaiden luonnollista poistumaa. Tutkimuksen päätavoite oli kuitenkin arvioida tietokannan kasvunopeuden ylärajaa, ja poistuma vaikuttaa siihen alentavasti.

5.1 Vastaukset tutkimuskysymyksiin

Johdannossa esitettyihin 4 tutkimuskysymykseen saatiin tutkimuksen perusteella seuraavat vastaukset.

1. Mitä tietokantaratkaisuja Movenium Oy:n tarpeisiin on tarjolla?

Tässä työssä tarkasteltiin seuraavia tietokantoja: MySQL, PostgreSQL, Oracle Database, MongoDB sekä Cassandra. Näistä Movenium Oy käyttää tällä hetkellä MySQL-tietokantaa. Muut tutkitut relaatiotietokannat, PostgreSQL ja Oracle Database, eivät tutkimuksen mukaan toisi riittäviä parannuksia MySQL:ään nähden, jotta vaihtaminen olisi perusteltua. PostgeSQL tarjoaisi paremman kirjoitusnopeuden, mutta Movenium Oy:n palvelut eivät sitä vaadi. Oracle Database:n tarjoama tuotetuki voisi tuoda lisäarvoa Movenium Oy:lle, mutta toisaalta Movenium Oy käyttää laajasti myös muita avoimeen lähdekoodiin perustuvia ohjelmia, kuten linux:ia[39] ja apache:a[40].

Tutkitut NoSQL tietokannat, MongoDB ja Cassandra, tarjoaisivat parempaa suorituskykyä sekä parempaa skaalautuvuutta. Ongelma näiden tietokantojen kanssa on vaikea käytön aloittaminen. NoSQL-tietokantojen datamalli on niin erilainen kuin relaatiotietokantojen, että ohjelmistot pitäisi ohjelmoida lähes alusta asti uudelleen.

[39] Linux on avoimeen lähdekoodiin perustuva käyttöjärjestelmä. http://www.ubuntu-fi.org/.
[40] Apache on avoimeen lähdekoodiin perustuva http-palvelinohjelma. http://httpd.apache.org/.

2. **Riittääkö Collectorin suorituskyky tulevina vuosina Movenium Oy:n tarpeisiin?**

Tässä työssä tehdyn tutkimuksen perusteella ei lähivuosina (2015-2018) ole näköpiirissä, että Collectorin suorituskyky ei olisi riittävä työajanseurantapalvelun tarpeisiin. Päätelmään päästään kolmea asiaa tarkastelemalla. 1) Tietokannan suorituskyky ei tutkimusten mukaan alene ainakaan alle 100 Gt kokoisena. 2) Jos yritys saa uusia asiakkaita arvoidulla nopeudella, ei tietokannan koko ylitä kyseistä arvoa ennen vuotta 2018. 3) Tutkimuksessa ei otettu kantaa palvelimien kykyyn käsitellä suuria määriä samanaikaisia pyyntöjä. Pyyntöjen lukumäärä voidaan kuitenkin pitää halutulla tasolla lisäämällä replikoitujen tietokantapalvelimien lukumäärää.

3. **Onko Collector paras valinta Movenium Oy:n tietokantaratkaisuksi?**

Tässä työssä tehdyn tutkimuksen perusteella ei löydetty syitä, jotka estäisivät Collectorin käytön lähivuosina (2015-2018). Muihin tietokantaratkaisuihin siirtyminen ei siis näyttäisi olevan välttämätöntä. Vastaus kysymykseen on siis, että ratkaisua ei tarvitse lähteä lyhyellä aikavälillä vaihtamaan.

4. **Mitä muutoksia Movenium Oy:n olemassa oleviin tuotteisiin tarvitaan suorituskyvyn varmistamiseksi?**

Tuotannosta kopioitua tietokantaa vasten palvelua testaamalla löydettiin järjestelmästä kuusi kohtaa, jotka vaativat korjaamista. Nämä on lueteltu kappaleessa 4.2.1.1. Kappaleessa on kerrottu myös korjausehdotukset kyseisiin ongelmiin. Näiden lisäksi Movenium Oy:n tulisi painottaa päätelaitteilla suoritettavien ohjelmien kehittämistä. Tällä tavoin palvelimien kuormaa saadaan siirrettyä loppukäyttäjien päätelaitteisiin.

5.2 Jatkotutkimustarpeet

Jatkotutkimusta tarvitaan Collector-järjestelmän kustannustehokkuudesta. Tässä työssä tutkittiin ainoastaan Collectorin suorituskykyä, eikä otettu kantaa siihen miten kalliiksi järjestelmän kehittäminen ja ylläpitäminen tulevat. Myöskään Collector-järjestelmän tarvitsemien palvelimien kustannuksia ei arvioitu. On mahdollista, että esimerkiksi DBaaS-tyyppiset palvelut tulisivat Movenium Oy:lle edullisemmaksi.

Lisää tutkimusta tarvitaan myös järjestelmän kuormituskestävyydestä. Tässä työssä tutkittiin miten kyselyiden suoritusajat kasvavan datamäärän kasvaessa, mutta työssä ei tutkittu miten suuria käyttäjämääriä palvelimet pystyvät samanaikaisesti palvelemaan. Tämän asian tutkiminen edellyttää täysin toisenlaista lähestymistapaa, missä palvelimia kuormitetaan siihen suunniteltujen ohjelmistojen avulla.

Viitteet

[1] A. Gupta, V. Harinarayan ja A. Rajaraman. Virtual Database Technology. IEEE. In, 14th International Conference on Data Engineering [konferenssijulkaisu]. 1998. s. 297–301. DOI:10.1109/ICDE.1998.655791.

[2] Dan Ma. The Business Model of 'Software-As-A-Service'. IEEE. In IEEE International Conference on Services Computing [konferenssijulkaisu]. 2007. s. 701–702. DOI:10.1109/SCC.2007.118.

[3] V. Choudhary. Software as a Service: Implications for Investment in Software Development. IEEE. In 40th Annual Hawaii International Conference on System Sciences [konferenssijulkaisu]. 2007. DOI:10.1109/HICSS.2007.493.

[4] Kang Sungjoo, Sungwon Kang ja S. Hur. A Design of the Conceptual Architecture for a Multitenant SaaS Application Platform. IEEE. In 2011 First ACIS/JNU International Conference on Computers, Networks, Systems and Industrial Engineering (CNSI) [konferenssijulkaisu]. 2011. s. 462–467. DOI:10.1109/CNSI.2011.56.

[5] Noel Rappin. Professional Ruby on Rails. Wiley. 2008. ISBN: 9780470223888.

[6] Bruce A. Tate ja Curt Hibbs. Ruby on Rails: Up and Running. O'Reilly Media, Inc. 2006. ISBN:978-0-596-52200-1.

[7] Sam Lightstone, Tom Nadeau ja Toby Teorey. Database Modeling and Design: Logical Design. Morgan Kaufmann. 2005. ISBN: 9780080470771.

[8] Xml technology. World Wide Web Consortium. http://www.w3.org/standards/xml/. Viitattu: 16.2.2014.

[9] Fawcett, Joe Ayers, Danny Quin ja Liam R.E. Beginning XML. Wiley. 2012. ISBN: 9781118162132.

[10] Julie Meloni. PHP Essentials. Course Technology PTR. 2003. ISBN: 9781931841344.

[11] Hong He. Applications Deployment on the SaaS Platform. IEEE. 5th International Conference on Pervasive Computing and Applications (ICPCA) [konferenssijulkaisu]. 2010. s. 232–237. DOI: 10.1109/ICPCA.2010.5704104.

[12] Yu-Hui Wang. The Role of SaaS Privacy and Security Compliance for Continued SaaS Use. IEEE. 7th International Conference on Networked Computing and Advanced Information Management (NCM) [konferenssijulkaisu]. 2011. s. 303–306.

[13] Eric Marks. Cloud Computing. John Wiley & Sons. 2010. ISBN: 9780470630495.

[14] Vijay Krishna Pallaw. Database Management Systems. Global Media Publications. 2010. ISBN: 9789350436592.

[15] Python Datastore Index Configuration. [Google App Engine dokumentaatio]. https://developers.google.com/appengine/docs/python/config/indexconfig. Viitattu: 10.9.2013.

[16] Juan Cáceres ja Luis M. Vaquero. Service Scalability Over the Cloud. Springer US. 2010. DOI: 10.1007/978-1-4419-6524-0_15.

[17] Wiesmann M. ja A. Schiper. Comparison of Database Replication Techniques Based on Total Order Broadcast. IEEE. IEEE Transactions on Knowledge and Data Engineering 17, no. 4 [konferenssijulkaisu]. 2005. s. 551–566. DOI:10.1109/TKDE.2005.54.

[18] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme ja G. Alonso. Database Replication Techniques: A Three Parameter Classification. IEEE. In Proceedings The 19th IEEE Symposium on Reliable Distributed Systems [konferenssijulkaisu]. 2000. s. 206–215. DOI:10.1109/RELDI.2000.885408.

[19] J. Balasubramanian, D.C. Schmidt, L. Dowdy ja O. Othman. Evaluating the Performance of Middleware Load Balancing Strategies. IEEE. In Enterprise Distributed Object Computing Conference [konferenssijulkaisu]. 2004. s. 135–146. DOI:10.1109/EDOC.2004.1342511.

[20] Shaoquan Fang ja Qiuli Tong. A Comparison of Multi-tenant Data Storage Solutions for Software-as-a-Service. IEEE. 6th International Conference on Computer Science Education (ICCSE) [konferenssijulkaisu]. 2011. s. 95–98. DOI: 10.1109/ICCSE.2011.6028592.

[21] Robert Sheldon. Beginning MySQL. John Wiley & Sons. 2005. ISBN: 9780764596575.

[22] Curioso. Andrew. Expert PHP and MySQL. Wrox. 2010. ISBN: 9780470643075.

[23] InnoDB and FOREIGN KEY Constraints. Oracle. http://dev.mysql.com/doc/refman/5.6/en/innodb-foreign-key-constraints.html. Viitattu: 26.2.2014.

[24] R. Kataoka, T. Satoh ja U. Inoue. A Multiversion Concurrency Control Algorithm for Concurrent Execution of Partial Update and Bulk Retrieval Transactions. IEEE. In , Tenth Annual International Phoenix Conference on Computers and Communications. [konferenssijulkaisu]. 1991. s. 130–136. DOI:10.1109/PCCC.1991.113802.

[25] Dan R. K. Ports ja Kevin Grittner. Serializable Snapshot Isolation in PostgreSQL. 2011. http://drkp.net/papers/ssi-vldb12.pdf. Viitattu: 26.2.2014.

[26] Tudorica B.G. ja C. Bucur. A Comparison between Several NoSQL Databases with Comments and Notes. IEEE. In Roedunet International Conference (RoEduNet), 2011 10th [konferenssijulkaisu]. 2011. DOI:10.1109/RoEduNet.2011.5993686.

[27] Lith, Adam ja Jakob Mattson. Investigating storage solutions for large data: A comparison of well performing and scalable data storage solutions for real time extraction and batch insertion of data. Göteborg: Department of Computer Science and Engineering, Chalmers University of Technology. 2010. http://publications.lib.chalmers.se/records/fulltext/123839.pdf. Viitattu: 6.3.2014.

[28] Chang Fay, Dean Jeffrey, Ghemawat Sanjay ja Hsieh Wilson C. Bigtable: A Distributed Storage System for Structured Data [Googlen tutkimus]. http://static.googleusercontent.com/media/research.google.com/fi//archive/bigtable-osdi06.pdf. Viitattu: 6.3.2014.

[29] Gansen Zhao, Weichai Huang, Shunlin Liang ja Yong Tang. Modeling MongoDB with Relational Model. IEEE. Fourth International Conference on Emerging Intelligent Data and Web Technologies (EIDWT) [konferenssijulkaisu]. 2013. s. 115–121. DOI: 10.1109/EIDWT.2013.25.

[30] Guoxi Wang ja Tang Jianfeng. The NoSQL Principles and Basic Application of Cassandra Model. IEEE. International Conference on Computer Science Service System (CSSS) [konferenssijulkaisu]. 2012. s. 1332–1335. DOI: 10.1109/CSSS.2012.336.

[31] Shashank Tiwari. Professional NoSQL. Wrox Press. 2011. ISBN: 9781118167816.

[32] Jing Han, Guan Le E. Haihong ja Jian Du. Survey on NoSQL Database. IEEE. 6th International Conference on Pervasive Computing and Applications (ICPCA) [konferenssijulkaisu]. 2011. s. 363–366. DOI: 10.1109/ICPCA.2011.6106531.

[33] Li, Yishan, ja S. Manoharan. A Performance Comparison of SQL and NoSQL Databases. IEEE. In 2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM) [konferenssijulkaisu]. 2013. s. 15–19. DOI:10.1109/PACRIM.2013.6625441.

[34] L. Okman, N. Gal-Oz, Y. Gonen, E. Gudes ja J. Abramov. Security Issues in NoSQL Databases. IEEE. IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom) [konferenssijulkaisu]. 2011. s. 541–547. DOI: 10.1109/TrustCom.2011.70.

[35] R. Rivest. The md5 message-digest algorithm. MIT Laboratory for Computer Science and RSA Data Security, Inc. 1992. https://tools.ietf.org/html/rfc1321. Viitattu: 26.2.2014.

[36] Kumar H., S. Kumar, R. Joseph, D. Kumar, S.K.S. Singh, A. Kumar ja P. Kumar. Rainbow Table to Crack Password Using MD5 Hashing Algorithm. IEEE. In 2013 IEEE Conference on Information Communication Technologies (ICT) [konferenssijulkaisu]. 2013. s. 433–439. DOI:10.1109/CICT.2013.6558135.

[37] P. Gauravaram. Security Analysis of Salt||password Hashes. IEEE. In 2012 International Conference on Advanced Computer Science Applications and Technologies (ACSAT) [konferenssijulkaisu]. 2012. s. 25–30. DOI:10.1109/ACSAT.2012.49.

[38] W. Lehner ja K.-U. Sattler. Database as a Service (DBaaS). IEEE. IEEE 26th International Conference on Data Engineering (ICDE) [konferenssijulkaisu]. 2010. s. 1216–1217. DOI: 10.1109/ICDE.2010.5447723.

[39] Damien Johnson. DBaaS (Database-as-a-Service) is Next – Google Unleashes Cloud Datastore [lehtiartikkeli]. http://www.highlightpress.com/dbaas-database-as-a-service-is-next-google-unleashes-cloud-datastore/1428/djohnson. Viitattu: 9.9.2013.

[40] Brian Proffitt. FoundationDB's NoSQL Breakthrough Challenges Relational Database Dominance. 2013. http://readwrite.com/2013/03/08/foundationdbs-nosql-breakthrough-challenges-relational-database-dominance. Viitattu: 6.3.2014.

[41] Benson Edward. The Art of Rails. Wiley. 2008. ISBN: 9780470386071.

[42] Rappin Noel. Professional Ruby on Rails. Wiley. 2008. ISBN: 9780470334300.

[43] V Viswanathan. Rapid Web Application Development: A Ruby on Rails Tutorial. IEEE Software 25, no. 6 [verkkolehti]. 2008. DOI: 10.1109/MS.2008.156.

[44] Steven Holzner. Beginning Ruby on Rails. Springer. 2007. ISBN: 9780470121085.

[45] Maras J., M. Stula, J. Carlson ja I. Crnkovic. Identifying Code of Individual Features in Client-Side Web Applications. IEEE. IEEE Transactions on Software Engineering 39, no. 12 [konferenssijulkaisu]. 2013. s. 1680–1697. DOI:10.1109/TSE.2013.38.

[46] Louis Columbus. IDC: 87% Of Connected Devices Sales By 2017 Will Be Tablets And Smartphones. Forbes [verkkolehti]. 2013. http://www.forbes.com/sites/louiscolumbus/2013/09/12/idc-87-of-connected-devices-by-2017-will-be-tablets-and-smartphones/. Viitattu: 28.2.2014.

[47] Ochin, Jugnu Gaur. Cross Browser Incompatibility: Reasons and Solutions. International Journal of Software Engineering & Applications (IJSEA), Vol.2, No.3 [verkkojulkaisu]. 2011. http://www.airccse.org/journal/ijsea/papers/0711ijsea05.pdf.

[48] Fernandes, J.L., I.C. Lopes, J.J.P.C. Rodrigues ja S. Ullah. Performance Evaluation of RESTful Web Services and AMQP Protocol. IEEE. In 2013 Fifth International Conference on Ubiquitous and Future Networks (ICUFN) [konferenssijulkaisu]. 2013. s. 810–815. DOI:10.1109/ICUFN.2013.6614932.

[49] David Herron. Node Web Development. Packt Publishing. 2011. ISBN: 9781849515153.

[50] Jim R. Wilson. Node.js the Right Way. The Pragmatic Programmers. 2013. ISBN: 978-1-937785-73-4.

[51] Good Phillip I. Introduction to Statistics Through Resampling Methods and R (2nd Edition). Wiley. 2013. ISBN: 9781118497562.

[52] Peter Zaitsev. Using delayed JOIN to optimize count(*) and LIMIT queries. MySQL Performance Blog. 2007. http://www.mysqlperformanceblog.com/2007/04/06/using-delayed-join-to-optimize-count-and-limit-queries/. Viitattu 24.2.2013.

[53] Samanta, D. Classic Data Structures. PHI Learning Pvt. Ltd. 2004. ISBN: 978-81-203-1874-8.

Liite: ohjelmakoodi tietokannan tutkimiseen

Alla on esitetty lähdekoodi jota käytettiin testidatan generoimiseen tietokantaan, sekä kyselyiden suoritusnopeuden mittaamiseen.

```
<?
// initilize collector
include("include.php");

// how many worktimes to add per day
$worktimes_per_day = 30;
$start_date = "2013-01-01";
$loop_days_count = 5000;
// set bulkdata parameters
$bulk_data = array("worktimes" => 100000, "users" => 100, "projects"
=> 1000);

$c = 0;

$worktime_data = array('user' => 229272,
   'project' => 229346,    'starttime'=> '12:00',
   'endtime' => '14:00',   'comment'  => "123456789012345",
   'addon1'  => "123456789012345",'addon2'=> "123456789012345",
   'addon3'  => "123456789012345",'addon4'=> "12345",
   'addon5'  => "12345",          'addon6'  => "12345",
   'addon7'  => "12345",          'addon8'  => "12345");

for ($i = 0; $i < $loop_days_count; $i++) {
    $insert_took = array();
    for ($j = 0; $j < $worktimes_per_day; $j++) {

        $worktime_data['date'] = date("Y-m-d", strtotime($start_date."
+ $i days"));
        $starttime = microtime(true);
        $data2->cadd("worktime", $worktime_data);
        // maesure inserting time
        $insert_took[] = microtime(true) - $starttime    }
        // add 100 000 bulk worktimes to database
```

```
        foreach ($bulk_data as $k => $v) add_bulk_data($k, $v);


        $worktime_count = $c + $bulk_data['worktimes'] * $i;
       log_time($worktime_count,measure_select_speed(),
'benchmark.txt');
       log_time($worktime_count,measure_select_speed_week_sum(),
'benchmark_week_sum.txt');
    log_time($worktime_count,measure_select_speed_old_get_function(),
'benchmark_old_get_function.txt');
    log_time($worktime_count,get_average($insert_took),
'benchmark_inserts.txt');
}


print "done";


// highspeed bulkdata adding function
function add_bulk_data($table, $count) {
   $query = "INSERT INTO data_rows (partnerid, formid, datetime,
userid,".
     "status) VALUES
     ".substr(str_repeat("(3,4,'1970-01-01 00:00:00',5,'normal'),",
$count),0,-1);
   mysql_query($query) or die(mysql_error());


   $values = array("data_datetime" => '1970-01-01 00:00:00',
"data_link" => 4,
                   "data_float" => 12345, "data_varchar" =>
"1234567890");


   if ($table == "worktimes") $other_rows = array("data_datetime" =>
3,
                "data_link" => 2, "data_float" => 3, "data_varchar" =>
2);
   if ($table == "users") $other_rows = array("data_varchar" => 4);
   if ($table == "projects") $other_rows = array("data_varchar" => 2);


   foreach ($other_rows as $t => $multiplier) {
        $query = "INSERT INTO ".$t." (value, schemaid, rowid) VALUES
          ".substr(str_repeat("('".$values[$t]."',5,6),", $count *
$multiplier),0,-1);
        mysql_query($query) or die(mysql_error());
   }
}
```

```
// measure time for fetching one months worktimes from database
function measure_select_speed() {
        global $data2;
        for ($i=0; $i < 5; $i++) {
                    mysql_query("RESET QUERY CACHE");
            $starttime = microtime(true);
            $result = $data2->cget("worktime",array("date" =>
                "{col} BETWEEN '2013-01-01' AND '2013-02-01'"),
array("limit" => "0,100"));
            $results[] = microtime(true) - $starttime;
        }

        return get_average($results);
}

// measure time for fetching one weeks worktimes summed by employee
from database
function measure_select_speed_week_sum() {
        global $data2;
        for ($i=0; $i < 5; $i++) {
                    mysql_query("RESET QUERY CACHE");
            $starttime = microtime(true);
            $data2->debug_mode = true;
            $result = $data2->cget("worktime",array("date" =>
                "{col} BETWEEN '2014-01-01' AND '2014-12-
31'"),array("group" => array("user")));
            if ($i == 0) var_dump($result);
            $results[] = microtime(true) - $starttime;
        }

        return get_average($results);
}

// measure time for fetching one months worktimes from database (old
function)
function measure_select_speed_old_get_function() {
        $data = new data();
        for ($i=0; $i < 5; $i++) {
                    mysql_query("RESET QUERY CACHE");
```

```
            $starttime = microtime(true);
            $result = $data->cget("worktime",array(
                "{date} BETWEEN '2013-01-01' AND '2013-02-01'"),
array("limit" => "0,100"));
            $results[] = microtime(true) - $starttime;
        }
        return get_average($results);
}

// get average of given results. (remove biggest and smallest)
function get_average($results) {
          $min = 1000000;
          $max = -1000000;
          foreach ($results as $k => $v) {
                    if ($v > $max) {$max_k = $k; $max = $v; }
                    if ($v < $min) {$min_k = $k; $min = $v; }
          }

          foreach ($results as $k => $v) {
                    if ($k != $min_k && $k != $max_k) {
                              $temp += $v;
                              $count++;
                    }
          }
          return $temp/$count;
}

// log time to harddrive
function log_time($counter, $time, $file = 'benchmark_.txt') {
          print $counter.": ".$time."\n";
          file_put_contents($file, "$counter\t".
                    ($time*1000)."\n", FILE_APPEND);
}

?>
```